



第2版

C++ 项目开发全程实录

10个完整开发项目，84集同步微视频

C++在线开发资源库，项目开发快用思维导图

明日科技 编著

140小时在线课程 实例资源库 模块资源库 项目资源库

源码资源库 面试资源库 测试题库 PPT电子课件 在线服务

清華大学出版社

软件项目开发全程实录

C++项目开发全程实录（第2版）

明日科技　编著

清華大学出版社
北　京

内 容 简 介

《C++项目开发全程实录（第 2 版）》以图书管理系统、餐饮管理系统、客房管理系统、人事考勤管理系统、商品采购管理系统、文档管理系统、FTP 管理系统、媒体播放器、吃豆子游戏和快乐五子棋等 10 个实际项目开发程序为案例，从软件工程的角度出发，按照项目的开发顺序，系统、全面地介绍了程序开发流程。从开发背景、需求分析、系统功能分析、业务流程图、数据库分析、数据库建模到系统开发，每一过程都作了详细的介绍。

本书及资源包特色包括 10 套项目开发完整案例，项目开发案例的同步视频和其源程序。登录网站还可获取各类资源库（模块库、题库、素材库）等项目案例常用资源，网站还提供技术论坛支持等。

本书案例涉及的行业广泛，实用性非常强。通过本书的学习，读者可以了解各个行业的特点，能够针对某一行业进行软件开发，也可以通过资源包中提供的案例源代码和数据库进行二次开发，以减少开发系统所需要的时间。

图书在版编目（CIP）数据

C++项目开发全程实录/明日科技编著. —2 版. —北京：清华大学出版社，2018
（软件项目开发全程实录）
ISBN 978-7-302-49949-7

Ⅰ. ①C…　Ⅱ. ①明…　Ⅲ. ①C 语言-程序设计　Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字（2018）第 066123 号

责任编辑：贾小红
封面设计：刘　超
版式设计：魏　远
责任校对：张丽萍
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京密云胶印厂
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张：**29　　**字　　数：**776 千字
版　　次：2013 年 10 月第 1 版　2018 年 7 月第 2 版　　**印　　次：**2018 年 7 月第 1 次印刷
印　　数：1～3500
定　　价：79.80 元

产品编号：079131-01

前　言

Preface

编写目的与背景

众所周知，当前社会需求和高校课程设置严重脱节，一方面企业找不到可迅速上手的人才，另一方面大学生就业难。如果有一些面向工作应用的案例参考书，让大学生得以参考，并能亲手去做，势必能缓解这种矛盾。本书就是这样一本书：项目开发案例型的、面向工作应用的软件开发类图书。编写本书的首要目的就是架起让学生从学校走向社会的桥梁。

其次，本书以完成小型项目为目的，让学生切身感受到软件开发给工作带来的实实在在的用处和方便，并非只是枯燥的语法和陌生的术语，从而激发学生学习软件开发的兴趣，让学生变被动学习为自主自发学习。

再次，本书的项目开发案例过程完整，不但适合在学习软件开发时作为小型项目开发的参考书，而且可以作为毕业设计的案例参考书。

最后，丛书第 1 版于 2008 年出版，并于 2011 年和 2013 年进行了两次改版升级，因为编写细腻，易学实用，配备全程视频讲解等特点，备受读者瞩目，丛书累计销售 20 多万册，成为近年来最受欢迎的软件开发项目案例类丛书之一。

转眼 5 年已过，我们根据读者朋友的反馈，对丛书内容进行了优化和升级，进一步修正之前版本中的疏漏之处，并增加了大量的辅助学习资源，相信这套书一定能带给您惊喜！

本书特点

微视频讲解

对于初学者来说，视频讲解是最好的导师，它能够引导初学者快速入门，使初学者感受到编程的快乐和成就感，增强进一步学习的信心。鉴于此，本书为大部分章节都配备了视频讲解，使用手机扫描正文小节标题一侧的二维码，即可在线学习项目制作的全过程。同时，本书提供了程序配置使用说明的讲解视频，扫描封底的二维码即可进行学习。

典型案例

本书案例均从实际应用角度出发，应用了当前流行的技术，涉及的知识广泛，读者可以从每个案例中积累丰富的实战经验。

代码注释

为了便于读者阅读程序代码，书中的代码均提供了详细的注释，并且整齐地纵向排列，可使读者

快速领略笔者意图。

代码贴士

案例类书籍通常会包含大量的程序代码，冗长的代码往往令初学者望而生畏。为了方便读者阅读和理解代码，本书避免使用连续大篇幅的代码，将其分割为多个部分，并对重要的变量、方法和知识点设计了独具特色的代码贴士。

知识扩展

为了增加读者的编程经验和技巧，书中每个案例都标记有注意、技巧等提示信息，并且在每章中都提供有一项专题技术。

本书约定

由于篇幅有限，本书每章并不能逐一介绍案例中的各模块。笔者选择了基础和典型的模块进行介绍，对于功能重复的模块，由于技术、设计思路和实现过程基本相同，因此没有在书中体现。读者在学习过程中若有相关疑问，请登录本书官方网站。本书中涉及的功能模块在资源包中都附带有视频讲解，方便读者学习。

适合读者

本书适合作为计算机相关专业的大学生、软件开发相关求职者和爱好者的毕业设计和项目开发的参考书。

本书服务

为了给读者提供更为方便快捷的服务，读者可以登录本书官方网站（www.mingrisoft.com）或清华大学出版社网站（www.tup.com.cn），在对应图书页面下载本书资源包，也可加入企业QQ（4006751066）进行学习交流。学习本书时，请先扫描封底的二维码，即可学习书中的各类资源。

本书作者

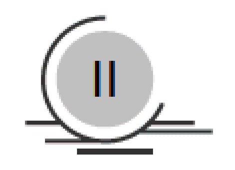
本书由明日科技软件开发团队组织编写，主要由李菁菁执笔，参与本书编写工作的还有赛奎春、王小科、周佳星、王国辉、杨丽、辛洪郁、张鑫、张宝华、申小琦、高春艳、葛忠月、刘杰、李磊、杨柳、赵宁、冯春龙、宋万勇、贾景波、吕玉翠、白宏健、隋妍妍、刘媛媛、李春林、何平、张云凯、申野、庞凤、胡冬、岳彩龙、潘建羽、张渤洋、梁英、于水晶、李雪、孙勃、卞昉、朱艳红、宋禹蒙、白兆松、依莹莹、李颖、王欢等，在此一并感谢！

在编写本书的过程中，我们本着科学、严谨的态度，力求精益求精，但错误、疏漏之处在所难免，敬请广大读者批评指正。

感谢您购买本书，希望本书能成为您的良师益友，成为您步入编程高手之路的踏脚石。

宝剑锋从磨砺出，梅花香自苦寒来。祝读书快乐！

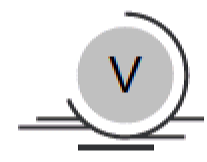
编　者

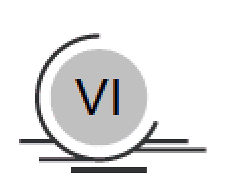


目　录

Contents









第 1 章

图书管理系统

（Visual C++ 6.0 实现）

随着现代社会的信息量不断增加，图书的种类及信息也越来越多，如何管理数量庞大的图书信息成为了图书管理工作中的一大难题。在计算机信息技术高速发展的今天，人们意识到原有的人工管理方式已经不能适应社会，而使用计算机信息系统来管理才是最有效率的一种手段。

通过学习本章，读者可以学到：

- 了解软件整体设计
- 掌握类的实际应用
- 掌握分页数据浏览
- 掌握文件存储数据



配置说明

1.1 开 发 背 景

随着现代图书市场竞争的愈演愈烈，如何以一种便捷的管理方式加快图书流通信息的反馈速度，降低图书库存占用，缩短资金周转时间，提高工作效率，已经成为能否增强图书企业竞争力的关键。信息技术的飞速发展给图书企业的管理带来了全新的变革，采用图书管理系统对图书企业的经营运作进行全程管理，不仅使企业摆脱了以往人工管理产生的一系列问题，而且使图书企业提高了管理效率，减少了管理成本，增加了经济效益。通过管理系统对图书企业的发展进行规划，可以收集大量关键、可靠的数据。企业决策层分析这些数据，作出合理决策，并及时调整，使之能够更好地遵循市场的销售规律，适应市场的变化，从而让企业能够在激烈的行业竞争中占据一席之地。

1.2 需 求 分 析

目前，图书市场的竞争日益激烈，这迫使图书企业希望采用一种新的管理方式来加快图书流通信息的反馈速度，而计算机信息技术的发展为图书管理注入了新的生机。通过对市场的调查得知，一款合格的图书信息管理系统必须具备以下 3 个特点。

☑ 能够对图书信息进行集中管理。
☑ 能够大大提高用户的工作效率。
☑ 能够对图书的部分信息进行查询。



1.3 系 统 设 计

1.3.1 系统目标

对于图书管理系统，必须要满足使用方便、操作灵活和安全性好等设计需求。设计本系统时应该完成以下几个目标。

☑ 图书的录入使用交互方式。
☑ 能够浏览文件中存储的全部图书。
☑ 图书信息在屏幕上的输出要有固定格式。
☑ 系统最大限度地实现易维护性和易操作性。
☑ 系统运行稳定、安全可靠。

1.3.2 系统功能结构

系统的功能结构如图 1.1 所示。

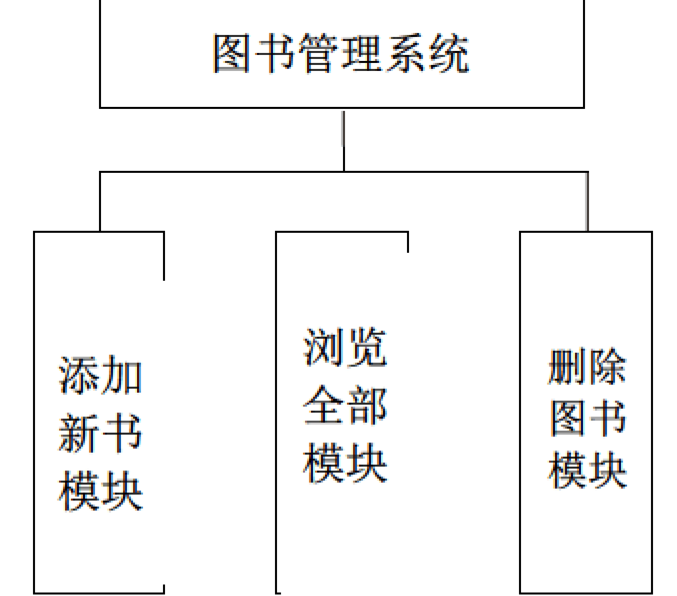
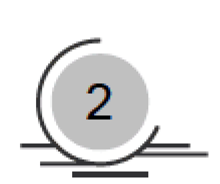


图 1.1 系统功能结构



☑ 添加新书模块：该模块主要供图书管理者使用。图书管理者应用该模块将图书信息录入到系统，系统将图书信息保存到文件中。
☑ 浏览全部模块：该模块供读者和图书管理者使用。图书管理者可以通过该模块查看图书是否存在，以及获取图书的编号，方便日后删除。读者可以根据该模块了解到图书的价格和作者等信息，从而决定是否购买。
☑ 删除图书模块：该模块主要供图书管理者使用。图书管理者可以通过该模块删除书店中已经销售完的图书的信息。

1.3.3　系统预览

图书管理系统由添加新书、浏览全部和删除图书 3 部分组成，由于篇幅有限，在此只给出部分功能预览图。

图书管理系统的主界面如图 1.2 所示。添加新书的界面如图 1.3 所示。

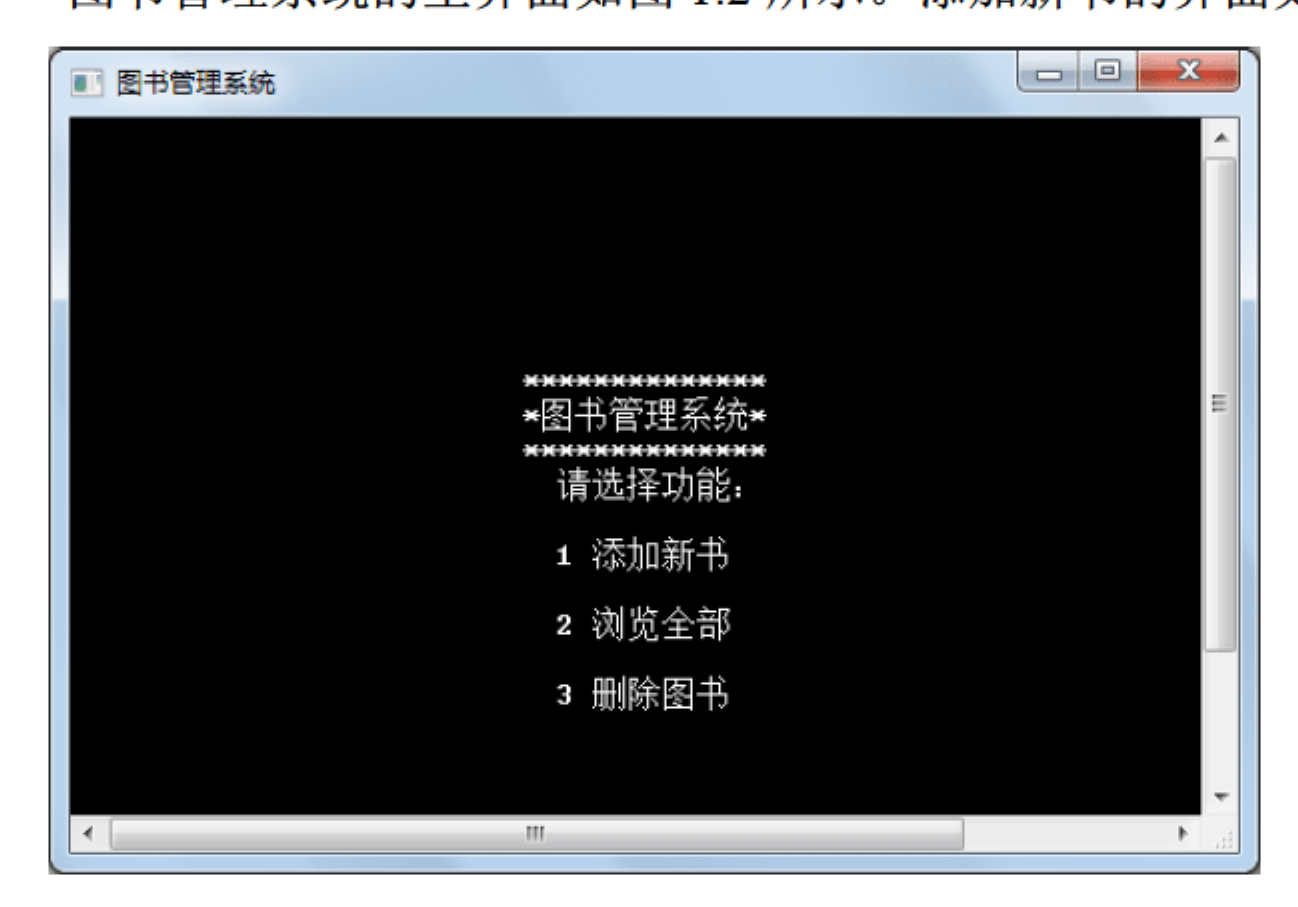


图 1.2　图书管理系统主界面

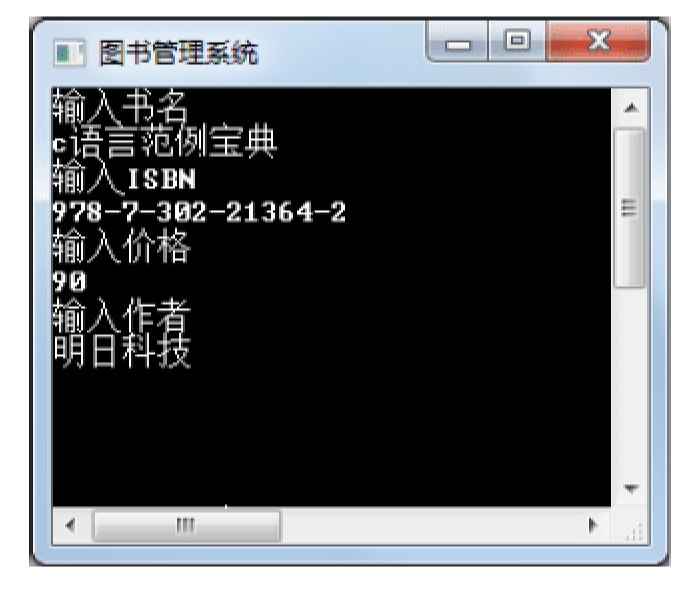


图 1.3　添加新书界面

浏览全部的界面如图 1.4 所示。



图 1.4　浏览全部界面

1.3.4　业务流程图

图书管理系统的业务流程图如图 1.5 所示。

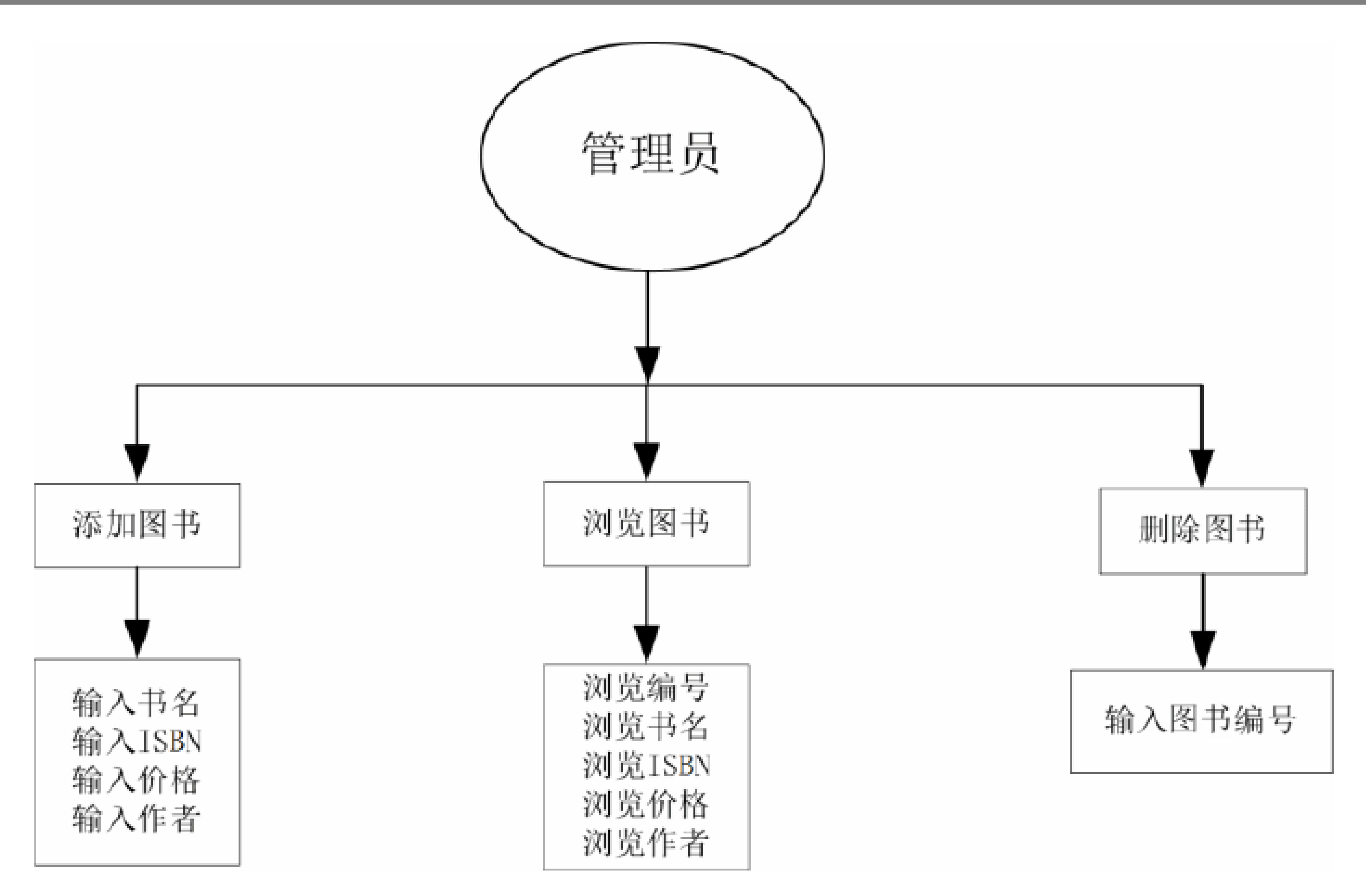


图 1.5　业务流程图



1.4　公共类设计

图书管理系统需要创建 CBook 类，通过 CBook 类可实现图书记录的写入和删除，还可以通过 CBook 类查看每条图书的信息。CBook 类中包含 m_cName、m_cIsbn、m_cPrice 和 m_cAuthor 共 4 个成员变量，分别代表图书的名称、ISBN 编号、价格和作者。在设计类时，可以将成员变量看作属性，此外，类中还需要有设置属性和获取属性的成员函数，设置属性的函数以 set 开头，获取属性的函数以 get 开头。CBook 类设计图如图 1.6 所示。

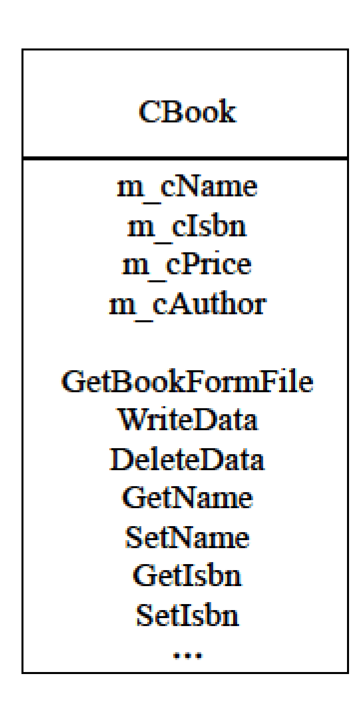


图 1.6　CBook 类设计图

CBook 类定义在头文件 Book.h 中，代码如下：

```
#define NUM1 128
#define NUM2 50
class CBook
{
public:
```

```
        CBook(){}
        CBook(char* cName,char* cIsbn,char* cPrice,char* cAuthor);
        ~CBook(){}
public:
        char* GetName();                    //获取图书名称
        void SetName(char* cName);          //设置图书名称
        char* GetIsbn();                    //获取图书 ISBN 编号
        void SetIsbn(char* cIsbn);          //设置图书 ISBN 编号
        char* GetPrice();                   //获取图书价格
        void SetPrice(char* cPrice);        //设置图书价格
        char* GetAuthor();                  //获取图书作者
        void SetAuthor(char* cAuthor);      //设置图书作者
        void WriteData();
        void DeleteData(int iCount);
        void GetBookFromFile(int iCount);
protected:
        char m_cName[NUM1];
        char m_cIsbn[NUM1];
        char m_cPrice[NUM2];
        char m_cAuthor[NUM2];

};
```

CBook 类成员函数的实现都存储在实现文件 Book.cpp 内。

```
#include "Book.h"
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
CBook::CBook(char* cName,char* cIsbn,char* cPrice,char* cAuthor)
{
        strncpy(m_cName,cName,NUM1);
        strncpy(m_cIsbn,cIsbn,NUM1);
        strncpy(m_cPrice,cPrice,NUM2);
        strncpy(m_cAuthor,cAuthor,NUM2);
}
char* CBook::GetName()
{
        return m_cName;
}
void CBook::SetName(char* cName)
{
        strncpy(m_cName,cName,NUM1);
}
char* CBook::GetIsbn()
{
        return m_cIsbn;
```

```
}
void CBook::SetIsbn(char* cIsbn)
{
    strncpy(m_cIsbn,cIsbn,NUM1);
}
char* CBook::GetPrice()
{
    return m_cPrice;
}
void CBook::SetPrice(char* cPrice)
{
    strncpy(m_cPrice,cPrice,NUM2);
}
char* CBook::GetAuthor()
{
    return m_cAuthor;
}
void CBook::SetAuthor(char* cAuthor)
{
    strncpy(m_cAuthor,cAuthor,NUM2);
}
```

函数 WriteData、GetBookFromFile 和 DeleteData 是类对象读写文件的函数，相当于操作数据库的接口。

（1）成员函数 WriteData 主要实现将图书对象写入到文件中。

```
void CBook::WriteData()
{
    ofstream ofile;
    ofile.open("book.dat",ios::binary|ios::app);
    try
    {
        ofile.write(m_cName,NUM1);
        ofile.write(m_cIsbn,NUM1);
        ofile.write(m_cPrice,NUM2);
        ofile.write(m_cAuthor,NUM2);
    }
    catch(...)
    {
        throw "file error occurred";
        ofile.close();
    }
    ofile.close();
}
```

（2）成员函数 GetBookFromFile 能够实现从文件中读取数据来构建对象。

```
void CBook::GetBookFromFile(int iCount)
{
```



```
        char cName[NUM1];
        char cIsbn[NUM1];
        char cPrice[NUM2];
        char cAuthor[NUM2];
        ifstream ifile;
        ifile.open("book.dat",ios::binary);
        try
        {
                ifile.seekg(iCount*(NUM1+NUM1+NUM2+NUM2),ios::beg);
                ifile.read(cName,NUM1);
                if(ifile.tellg()>0)
                        strncpy(m_cName,cName,NUM1);
                ifile.read(cIsbn,NUM1);
                if(ifile.tellg()>0)
                        strncpy(m_cIsbn,cIsbn,NUM1);
                ifile.read(cPrice,NUM2);
                if(ifile.tellg()>0)
                        strncpy(m_cIsbn,cIsbn,NUM2);
                ifile.read(cAuthor,NUM2);
                if(ifile.tellg()>0)
                        strncpy(m_cAuthor,cAuthor,NUM2);
        }
        catch(...)
        {
                throw "file error occurred";
                ifile.close();
        }
        ifile.close();
}
```

（3）成员函数 DeleteData 负责将图书信息从文件中删除。

```
void CBook::DeleteData(int iCount)
{
        long respos;
        int iDataCount=0;
        fstream file;
        fstream tmpfile;
        ofstream ofile;
        char cTempBuf[NUM1+NUM1+NUM2+NUM2];
        file.open("book.dat",ios::binary|ios::in|ios::out);
        tmpfile.open("temp.dat",ios::binary|ios::in|ios::out|ios::trunc);
        file.seekg(0,ios::end);
        respos=file.tellg();
        iDataCount=respos/(NUM1+NUM1+NUM2+NUM2);
        if(iCount < 0 && iCount > iDataCount)
        {
                throw "Input number error";
        }
        else
```

```
        {
            file.seekg((iCount)*(NUM1+NUM1+NUM2+NUM2),ios::beg);
            for(int j=0;j<(iDataCount-iCount);j++)
            {
                memset(cTempBuf,0,NUM1+NUM1+NUM2+NUM2);
                file.read(cTempBuf,NUM1+NUM1+NUM2+NUM2);
                tmpfile.write(cTempBuf,NUM1+NUM1+NUM2+NUM2);
            }
            file.close();
            tmpfile.seekg(0,ios::beg);
            ofile.open("book.dat");
            ofile.seekp((iCount-1)*(NUM1+NUM1+NUM2+NUM2),ios::beg);
            for(int i=0;i<(iDataCount-iCount);i++)
            {
                memset(cTempBuf,0,NUM1+NUM1+NUM2+NUM2);
                tmpfile.read(cTempBuf,NUM1+NUM1+NUM2+NUM2);
                ofile.write(cTempBuf,NUM1+NUM1+NUM2+NUM2);
            }
        }
        tmpfile.close();
        ofile.close();
        remove("temp.dat");
}
```



1.5 主窗体模块设计

1.5.1 主窗体模块概述

系统主程序界面是应用程序提供给用户访问其他功能模块的平台，根据实际需要，图书管理系统的主界面采用了传统的“数字选择功能”风格。输入数字 1 进入到添加新书模块，输入数字 2 进入到浏览全部模块，输入数字 3 进入到删除图书模块。图书管理系统的主界面如图 1.2 所示。

1.5.2 主窗体模块技术分析

要实现图书管理系统的功能，需要对引用库函数添加头文件引用。头文件引用和宏定义的代码如下：

```
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <fstream>
#include "Book.h"

#define CMD_COLS 80
```



```
#define CMD_LINES 25
using namespace std;
```

除主函数外，系统自定义了许多函数，主要函数及功能如下。

- ☑ void SetScreenGrid()：设置屏幕显示的行数和列数。
- ☑ void ClearScreen()：清除屏幕信息。
- ☑ void SetSysCaption(const char *pText)：设置窗体标题栏。
- ☑ void ShowWelcome()：显示欢迎信息。
- ☑ void ShowRootMenu()：显示开始菜单。
- ☑ void WaitView(int iCurPage)：浏览数据时等待用户操作。
- ☑ void WaitUser()：等待用户操作。
- ☑ void GuideInput()：使用向导添加图书信息。
- ☑ int GetSelect()：获得用户菜单选择。
- ☑ long GetFileLength(ifstream & ifs)：获取文件长度。
- ☑ void ViewData(int iSelPage)：浏览所有图书记录。
- ☑ void DeleteBookFromFile()：在文件中产生图书信息。
- ☑ void mainloop()：主循环。

1.5.3　主窗体模块实现过程

图书管理系统的主窗体设计实现过程如下。

（1）在控制台中输入 mode 命令可以设置控制显示信息的行数、列数和背景颜色等信息。SetScreenGrid 函数主要通过 system 函数来执行 mode 命令，CMD_COLS 和 CMD_LINES 是宏定义中的值。

```
void SetScreenGrid()
{
	char sysSetBuf[80];
	sprintf(sysSetBuf,"mode con cols=%d lines=%d",CMD_COLS,CMD_LINES);
	system(sysSetBuf);
}
```

（2）SetSysCaption 函数主要完成在控制台的标题栏上显示 Sample 信息。控制台的标题栏信息可以使用 title 命令来设置，函数中使用 system 函数来执行 title 命令。

```
void SetSysCaption()
{
	system("title Sample");
}
```

（3）ClearScreen 函数主要通过 system 函数来执行 cls 命令，完成控制台屏幕信息的清除。

```
void ClearScreen()
{
	system("cls");
}
```

（4）SetSysCaption 函数共有两个版本，这是 SetSysCaption 函数的另一个版本，主要实现在控制台的标题栏上显示指定字符。

```
void SetSysCaption( const char *pText)
{
	char sysSetBuf[80];
	sprintf(sysSetBuf,"title %s",pText);
	system(sysSetBuf);
}
```

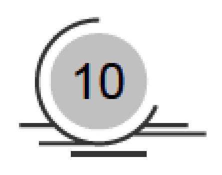
（5）ShowWelcome 函数在屏幕上显示“图书管理系统”字样的欢迎信息，“图书管理系统”字样应尽量显示在屏幕的中央位置。

```
void ShowWelcome()
{
	for(int i=0;i<7;i++)
	{
		cout << endl;
	}
	cout << setw(40);
	cout << "**************" << endl;
	cout << setw(40);
	cout << "图书管理系统" << endl;
	cout << setw(40);
	cout << "**************" << endl;
}
```

（6）ShowRootMenu 函数主要显示系统的主菜单，系统中有 3 个菜单选项，分别是添加新书、浏览全部和删除图书。3 个菜单选项是进入系统 3 个模块的入口。

```
void ShowRootMenu()
{
	cout << setw(40);
	cout << "请选择功能" << endl;
	cout << endl;
	cout << setw(38);
	cout << "1  添加新书" << endl;
	cout << endl;
	cout << setw(38);
	cout << "2  浏览全部" << endl;
	cout << endl;
	cout << setw(38);
	cout << "3  删除图书" << endl;
}
```

（7）WaitUser 函数主要负责当程序进入某一模块后，等待用户进行处理。用户可以选择返回主菜单，也可以直接退出系统。



```
void WaitUser()
{
	int iInputPage=0;
	cout << "enter·返回主菜单  q 退出" << endl;
	char buf[256];
	gets(buf);
	if(buf[0]=='q')
		system("exit");
}
```

（8）main 函数是程序的入口，主要调用了 SetScreenGrid、SetSysCaption 和 mainloop 3 个函数，其中，mainloop 函数是主函数，负责模块执行的调度，主要代码如下：

```
void mainloop()
{
	ShowWelcome();
	while(1)
	{
		ClearScreen();
		ShowWelcome();
		ShowRootMenu();
		switch(GetSelect())
		{
		case 1:
			ClearScreen();
			GuideInput();
			break;
		case 2:
			ClearScreen();
			ViewData();
			break;
		case 3:
			ClearScreen();
			DeleteBookFromFile();
			break;
		}
	}
}
```

（9）GetSelect 函数主要负责获取用户在菜单中的选择。

```
int GetSelect()
{
	char buf[256];
	gets(buf);
	return atoi(buf);
}
```

其他函数都应用在添加新书模块、浏览全部模块和删除图书模块中，相关内容将在具体模块中讲解。



1.6 添加新书模块设计

1.6.1 添加新书模块概述

在图书管理系统主窗体中输入数字 1，则进入到添加新书模块中。在添加新书模块中，用户需要输入所要添加的图书的书名、ISBN 编码、价格以及作者信息，其运行效果如图 1.7 所示。

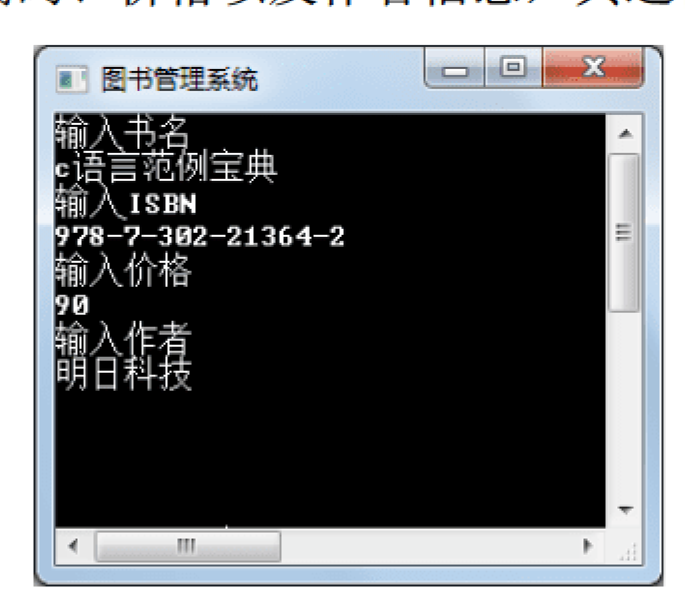


图 1.7 添加新书

1.6.2 添加新书模块技术分析

在添加新书模块中定义了 GuideInput 函数，通过在 main 函数中调用来完成添加图书的功能。

```
void GuideInput();
```

其次，利用 CBook 类构建一个 CBook 对象，通过 CBook 对象的成员函数 WriteData 将图书信息写入文件。

```
CBook book(inName,inIsbn,inPrice,inAuthor);
book.WriteData();
```

1.6.3 添加新书模块实现过程

图书管理系统中添加新书模块的实现代码如下：

```
void GuideInput()
{
    char inName[NUM1];
    char inIsbn[NUM1];
    char inPrice[NUM2];
    char inAuthor[NUM2];
```



```
        cout << "输入书名" << endl;
            cin >> inName;
        cout << "输入 ISBN" << endl;
            cin >> inIsbn;
        cout << "输入价格" << endl;
            cin >> inPrice;
        cout << "输入作者" << endl;
            cin >> inAuthor;
        CBook book(inName,inIsbn,inPrice,inAuthor);
        book.WriteData();
        cout << "Write Finish" << endl;
        WaitUser();
}
```

1.7　浏览全部模块设计



1.7.1　浏览全部模块概述

在图书管理系统主窗体中输入数字 2，则进入到浏览全部模块。该模块中可按页数显示图书记录，每页可以显示 20 条记录。主要实现的功能是显示所有图书的编号、图书名、ISBN 编码、价格以及作者信息，记录了当前记录中书的总数量、共有多少页及当前页数，还实现了翻页及返回主菜单的功能。其运行效果如图 1.8 所示。



图 1.8　浏览全部

1.7.2　浏览全部模块技术分析

图书管理系统中浏览全部模块主要通过定义函数 ViewData 来完成。在函数 ViewData 中直接使用文件流类打开存储图书信息的文件 book.dat。

```
void ViewData(int iSelPage = 1)
```

再定义一个 GetFileLength 函数，用来获取文件的长度。函数需要指定一个文件流对象，然后根据文件流的 tellg 函数计算出文件流绑定的文件长度。计算过程是先通过 tellg 函数获取文件指针的位置，然后通过 seekg 函数将文件指针移到文件末尾，再通过 tellg 函数获取文件指针的位置，此时的文件指

针的位置就是文件的长度，最后通过 seekg 函数将文件指针恢复到原来的位置。

```
long GetFileLength(ifstream & ifs)
```

1.7.3 浏览全部模块实现过程

在函数 ViewData 中直接使用文件流类打开存储图书信息的文件 book.dat，然后根据页序号读取文件内容，因为每条图书记录的长度相同，这样就很容易计算出每条记录在文件中的位置，然后将文件指针移动到每页第一条图书记录处，顺序地从文件中读取 20 条记录，并将信息显示在屏幕上。其代码如下：

```
void ViewData(int iSelPage = 1)
{
	int iPage=0;
	int iCurPage=0;
	int iDataCount=0;
	char inName[NUM1];                                          //存储图书名称的变量
	char inIsbn[NUM1];                                          //存储图书 ISBN 编号的变量
	char price[NUM2];                                           //存储图书价格的变量
	char inAuthor[NUM2];                                        //存储图书作者的变量
	bool bIndex=false;
	int iFileLength;
	iCurPage=iSelPage;
	ifstream ifile;
	ifile.open("book.dat",ios::binary);
	iFileLength=GetFileLength(ifile);
	iDataCount=iFileLength/(NUM1+NUM1+NUM2+NUM2)           //根据文件长度，计算文件中总的记录数
	if(iDataCount>=1)
		bIndex=true;
	iPage=iDataCount / 20+1;
	ClearScreen();                                              //清除屏幕信息
	cout << " 共有记录" << iDataCount <<" ";
	cout << " 共有页数" << iPage << " ";
	cout << " 当前页数" << iCurPage << " ";
	cout << " n 显示下一页 m 返回" << endl;
	cout << setw(5)<<"Index";
	cout << setw(22) << "Name" << setw(22) << "Isbn";
	cout << setw(15) << "Price" << setw(15) << "Author";
	cout << endl;
	try
	{
		//根据图书记录编号查找在文件中的位置
		ifile.seekg((iCurPage-1)*20*(NUM1+NUM1+NUM2+NUM2),ios::beg);
		if(!ifile.fail())
		{
			for(int i=1;i<21;i++)
			{
				memset(inName,0,128);                       //将变量清零
				memset(inIsbn,0,128);
```

```
                memset(price,0,50);
                memset(inAuthor,0,50);
                if(bIndex)
                    cout <<setw(3)<< ((iCurPage-1)*20+i);
                ifile.read(inName,NUM1);                    //读取图书名称
                cout <<setw(24)<< inName;
                ifile.read(inIsbn,NUM1);                    //读取图书 ISBN 编号
                cout <<setw(24)<< inIsbn;
                ifile.read(price,NUM2);                     //读取图书价格
                cout <<setw(12)<< price;
                ifile.read(inAuthor,NUM2);                  //读取图书作者
                cout <<setw(12)<< inAuthor;
                cout << endl;
                if(ifile.tellg()<0)
                    bIndex=false;
                else
                    bIndex=true;
                }
            }
        }
        catch(...)
        {
            cout << "throw file exception" << endl;
            throw "file error occurred";                    //抛出异常
            ifile.close();                                  //异常后关闭文件流
        }
        if(iCurPage<iPage)
        {
            iCurPage=iCurPage+1;
            WaitView(iCurPage);                             //等待用户处理
        }
        else
        {
            WaitView(iCurPage);                             //等待用户处理
        }
        ifile.close();
}
```

GetFileLength 函数的代码如下：

```
long GetFileLength(ifstream & ifs)
{
        long tmppos;
        long respos;
        tmppos=ifs.tellg();
        ifs.seekg(0,ios::end);
        respos=ifs.tellg();
        ifs.seekg(tmppos,ios::beg);
        return respos;
}
```



1.8 删除图书模块设计

1.8.1 删除图书模块概述

在图书管理系统的主窗体中输入数字 3，则进入到删除图书模块。在删除图书的模块中，通过输入想要删除的图书的顺序编号即可删除此图书，其效果如图 1.9 所示。

按图 1.9 所示操作，按 Enter 键之后返回到主窗体界面，再次选择浏览功能，删除图书后可以浏览如图 1.10 所示的全部图书内容，与图 1.8 比较，可以发现，编号为 1 的图书内容被删除。



图 1.9 删除图书



图 1.10 删除图书之后再次浏览全部图书

1.8.2 删除图书模块技术分析

在图书管理系统中，删除图书模块的设计主要是通过定义一个 DeleteBookFromFile 函数，并由 main 函数调用 DeleteBookFromFile 函数来完成的。

```
void DeleteBookFromFile()
```

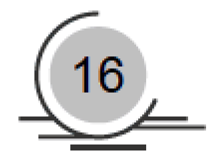
另外，在 DeleteBookFromFile 中调用 CBook 类的 DeleteData 成员函数。DeleteData 成员函数用于设置所删除图书在文件中的顺序编号，在浏览图书时可以看到此编号。

```
tmpbook.DeleteData(iDelCount);
    cout << "Delete Finish" << endl;
```

1.8.3 删除图书模块实现过程

在图书管理系统中，删除图书模块 DeleteBookFromFile 函数的实现代码如下：

```
void DeleteBookFromFile()
{
    int iDelCount;
    cout << "Input delete index" << endl;
    cin >> iDelCount;
    CBook tmpbook;
    tmpbook.DeleteData(iDelCount);
    cout << "Delete Finish" << endl;
    WaitUser();
```



```
}
void WaitView(int iCurPage)
{
	char buf[256];
	gets(buf);
	if(buf[0]=='q')
		system("exit");
	if(buf[0]=='m')
		mainloop();
	if(buf[0]=='n')
		ViewData(iCurPage);
}
```

1.9　实现全部模块



在 main.cpp 中添加以下代码，实现全部模块的主函数：

```
void main()
{

	SetScreenGrid();
	SetSysCaption("图书管理系统");
	mainloop();
}
```

1.10　项目文件清单

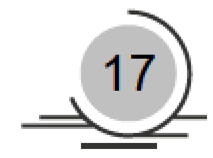
图书管理系统的文件清单如表 1.1 所示。

表 1.1　图书管理系统设计清单

文　件　名	文 件 类 型	说　　明
Book.cpp	源文件	实现图书记录的写入和删除
book.dat	数据文件	存放图书对象
Main.cpp	源文件	主窗体设计
Sample.dsw	工程文件	工程文件
Book.h	头文件	定义图书类

1.11　本 章 总 结

至此，一个简单的图书管理系统就完成了，本系统的开发过程符合软件工程方面的要求，但由于篇幅的关系，系统设计得比较小，没有应用模型，读者可以使用一些模型技术来不断完善该系统。



第 2 章

餐饮管理系统

（Visual C++ 6.0+Microsoft Access 2010 实现）

餐饮管理系统是饮食行业不可缺少的部分,它的内容对企业的决策和管理都至关重要，所以餐饮管理系统应能够为用户提供充足的信息和快捷的查询手段。但一直以来，人们使用的餐饮管理系统均是以人为主体的，需要很多的人力、物力，且效率不是很高，在系统运营时也可能产生人为的失误，以致餐饮管理工作既烦琐又不利于分析企业的经营状况。

作为计算机应用的一部分，使用计算机对餐饮信息进行管理，具有人工管理所无法比拟的优点，如统计结账快速、安全保密性好、可靠性高、存储量大、寿命长、成本低等。这些优点能够极大地提高餐饮管理的效率，增强企业的竞争力，同时也是企业科学化、正规化管理与世界接轨的重要条件。

通过学习本章，读者可以学到：

- **使用 Microsoft Access 2010 数据库**
- **使用 ADO 连接数据库**
- **通过 SQL 语句对数据库进行操作**
- **备份、还原数据库**
- **打包发行文件**



配置说明



2.1　开 发 背 景

俗话说：“民以食为天。”随着人民生活水平的提高，餐饮业在服务行业中的地位越来越重要。从激烈的竞争中脱颖而出，已成为每位餐饮业经营者所追求的目标。

经过多年发展，餐饮管理已经逐渐由人工管理进入到重视规范、科学管理的阶段。众所周知，在科学管理的具体实现方法中，最有效的就是应用管理软件进行管理。

以往的人工操作管理中存在着许多问题，例如：

☑　人工计算账单容易出现错误。

☑　收银工作中容易发生账单丢失。

☑　客人具体消费信息难以查询。

☑　无法对以往营业数据进行查询。

2.2　需 求 分 析

随着餐饮行业的迅速发展，现有的人工管理方式已不能满足工作需求。广大餐饮业经营者已经意识到使用计算机信息技术的重要性，决定采用计算机管理系统进行管理。

根据餐饮行业的特点和该企业的实际情况，该系统应以餐饮业务为基础，突出前台管理，从专业角度出发，提供科学、有效的管理模式。点菜方面采取表单加数据的方式使用户能直观地管理数据信息，并能有效地管理每个台号所点的酒菜。点菜收银管理可实现点菜、结账和清台功能。进货管理可记录商品入库情况。点菜收银、营业分析和库房管理的有机结合，可为确定酒店的经营方向提供依据，为其发展提供重要保证。

2.3　系 统 设 计



2.3.1　系统目标

餐饮管理系统将实现如下目标。

☑　减少前台服务人员的数量，减少经营者的人员开销。

☑　提高操作速度，提高顾客的满意程度。

☑　使经营者能够查询一些历史数据。

2.3.2　系统功能结构

餐饮管理系统包含前台服务、后台服务、财政服务和系统服务几部分功能，其功能结构如图 2.1 所示。

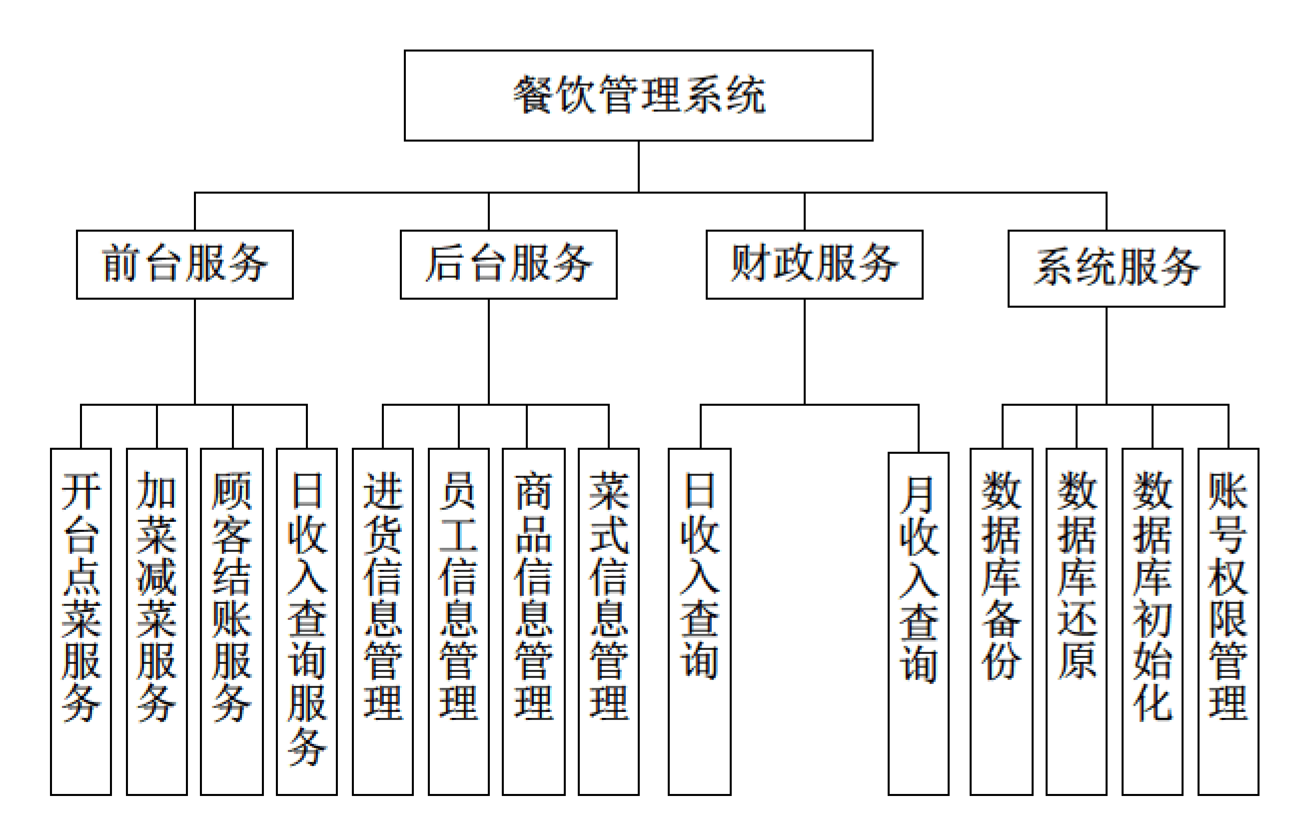


图 2.1　餐饮管理系统功能结构图

2.3.3　系统预览

餐饮管理系统由多个功能组成，下面仅列出几个典型的功能界面，其他界面可参见资源包中的源程序。典型的功能界面如图 2.2～图 2.5 所示。

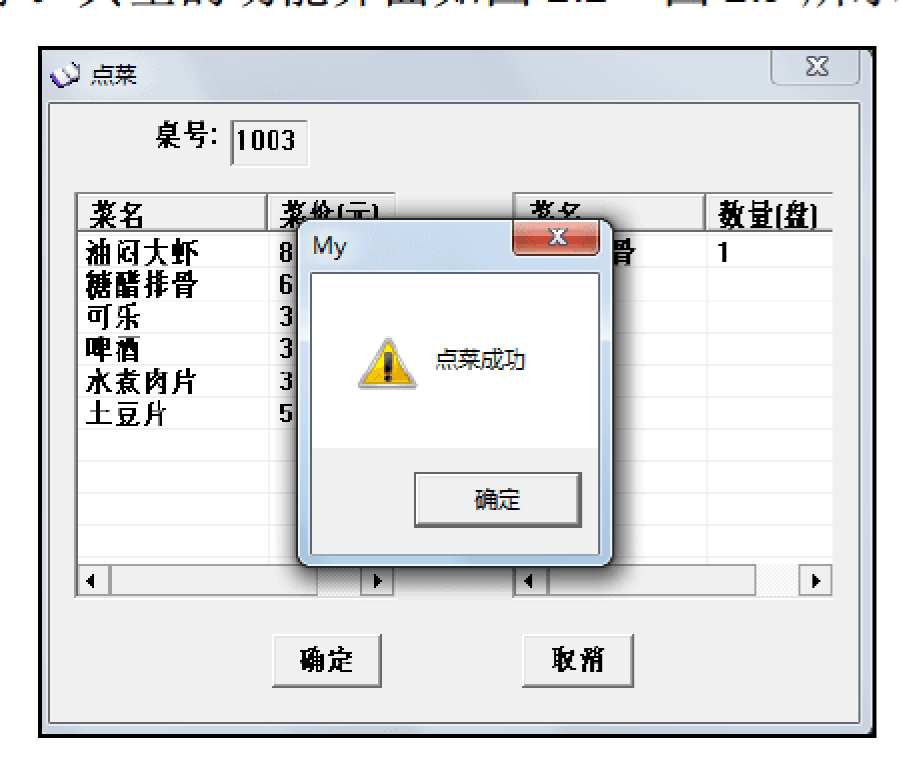


图 2.2　开台点菜服务界面



图 2.3　顾客结账服务界面

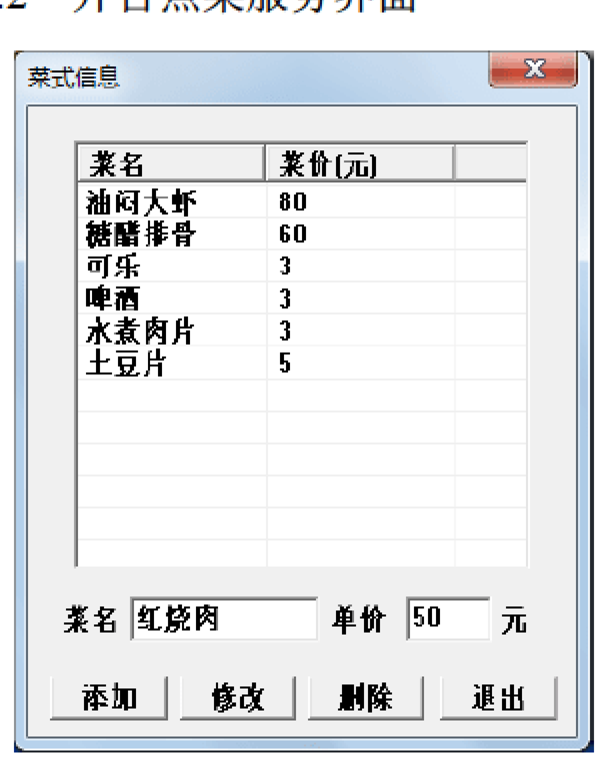
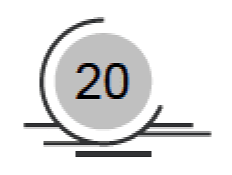


图 2.4　菜式信息管理界面



图 2.5　数据库还原和数据库备份界面





2.3.4　业务流程图

餐饮管理系统的业务流程图如图 2.6 所示。

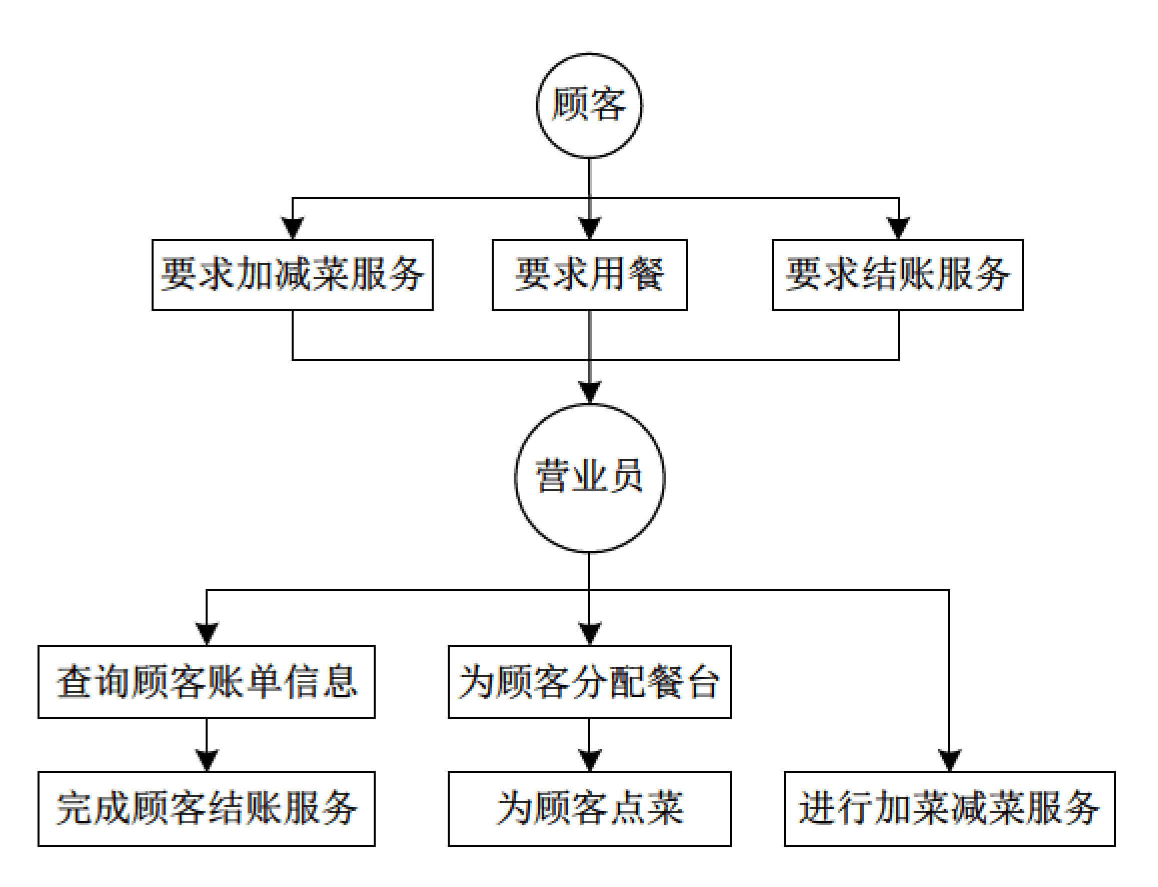


图 2.6　餐饮管理系统的业务流程图

2.3.5　数据库设计

一个好的数据库是每一个成功的系统必不可少的部分，数据库设计则是系统设计中最关键的一步。所以，要根据系统的信息量设计一个合适的数据库。

1．数据库分析

因为餐饮管理系统中需存储的数据信息量不大，对数据库的要求并不是很高，所以，本系统采用了 Microsoft Access 2010 数据库，数据库名称为 canyin。在数据库中一共建立了 7 张数据表，用于存储不同的信息，如图 2.7 所示。

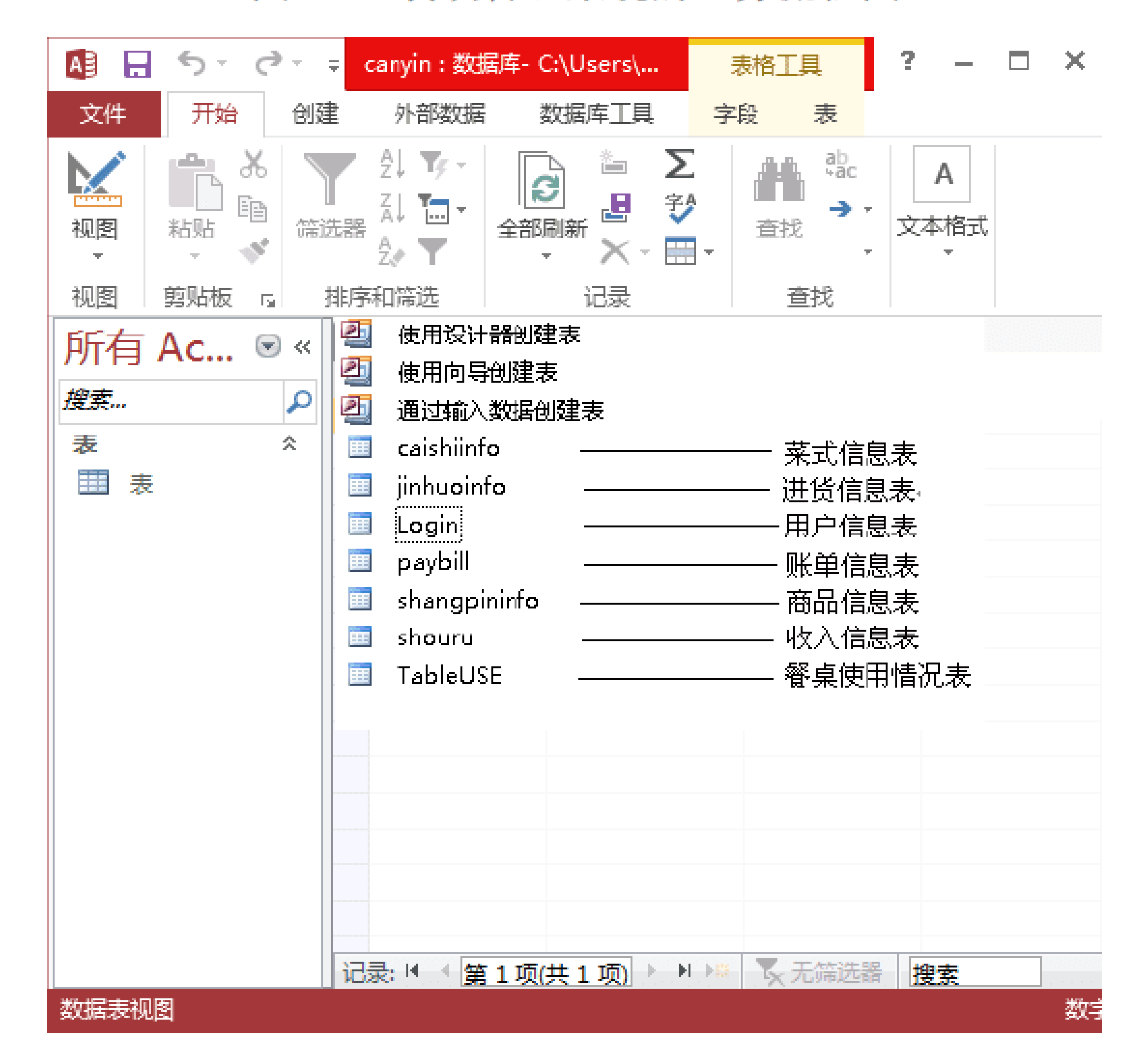


图 2.7　数据库 canyin 中的表

2．数据库概念设计

（1）用户信息实体

用户信息实体包括用户登录账号、用户登录密码和用户权限。用户信息实体 E-R 图如图 2.8 所示。

（2）菜式信息实体

菜式信息实体包括菜式名称和菜式价格。菜式信息实体 E-R 图如图 2.9 所示。

（3）进货信息实体

进货信息实体包括商品名称、商品价格、商品数量和进货时间。进货信息实体 E-R 图如图 2.10 所示。

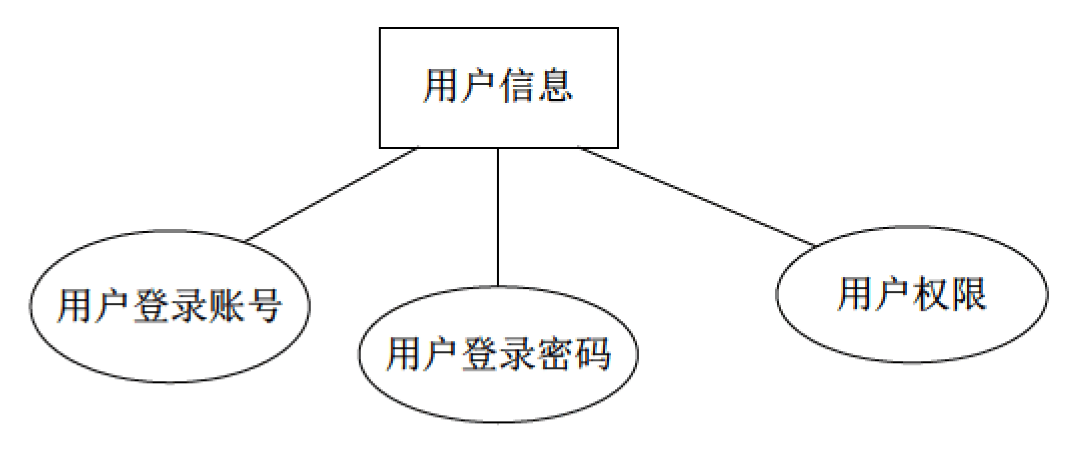


图 2.8　用户信息实体 E-R 图

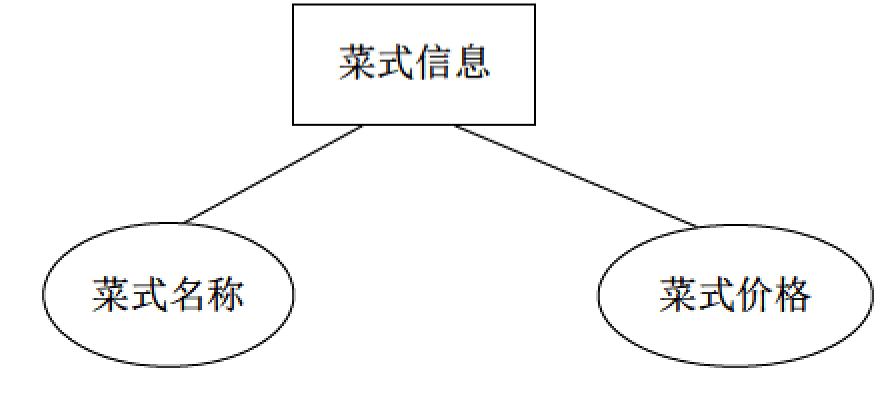


图 2.9　菜式信息实体 E-R 图

（4）账单信息实体

账单信息实体包括菜式名称、菜式价格、菜式数量和结账桌号。账单信息实体 E-R 图如图 2.11 所示。



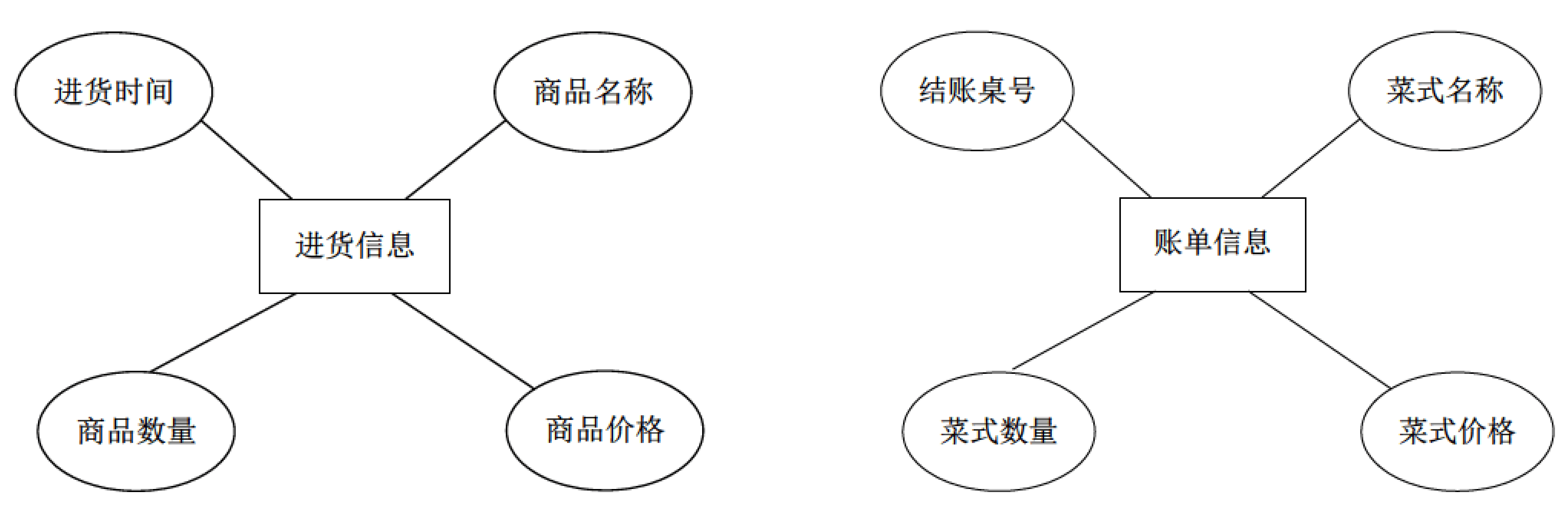


图 2.10　进货信息实体 E-R 图　　　图 2.11　账单信息实体 E-R 图

（5）商品信息实体

商品信息实体包括商品名称和商品单价。商品信息实体 E-R 图如图 2.12 所示。

（6）收入信息实体

收入信息实体包括日收入金额和收入时间。收入信息实体 E-R 图如图 2.13 所示。

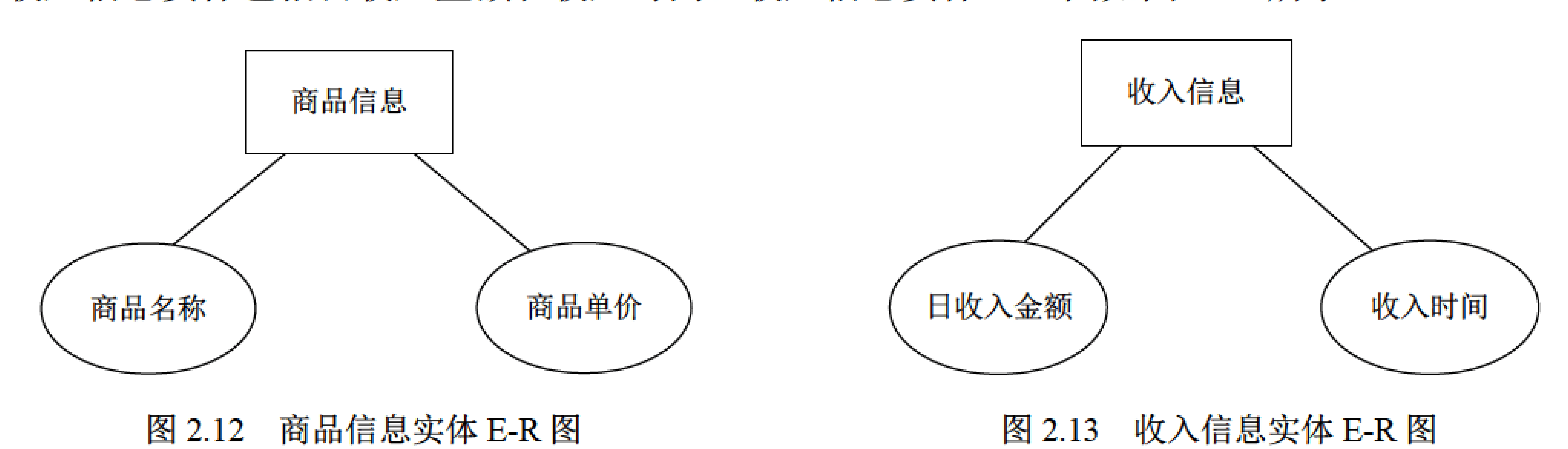


图 2.12　商品信息实体 E-R 图　　　图 2.13　收入信息实体 E-R 图

（7）餐桌使用情况实体

餐桌使用情况实体包括餐桌桌号和餐桌状态。餐桌使用情况实体 E-R 图如图 2.14 所示。

3．数据库逻辑结构设计

完成了上述实体 E-R 图，接下来就该创建数据表。下面以创建菜式信息表（caishiinfo）为例演示如何创建数据表。

（1）新建数据表

在数据库创建完毕之后，选择“视图”选项，然后再选择“设计视图”选项，将弹出如图 2.15 所示的对话框，提示用户输入新建表的名称。

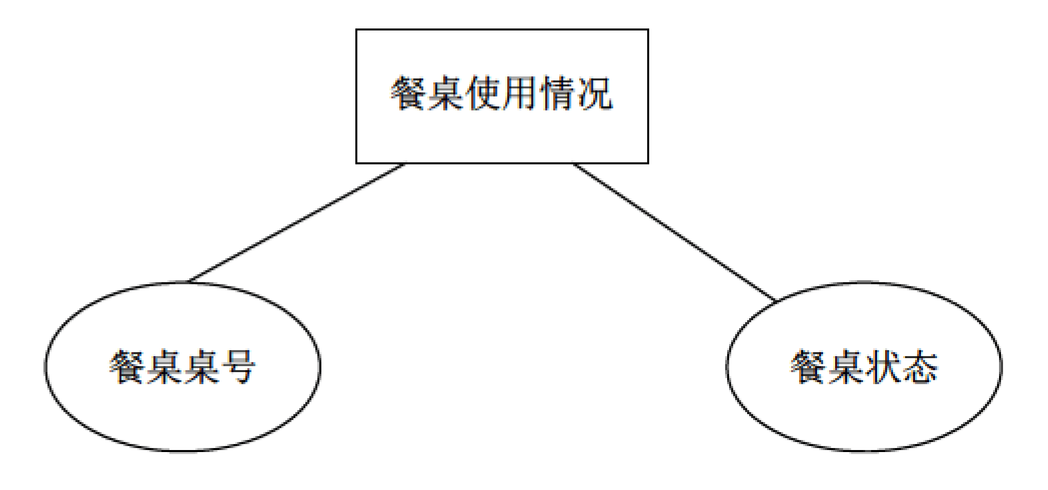


图 2.14　餐桌使用情况实体 E-R 图

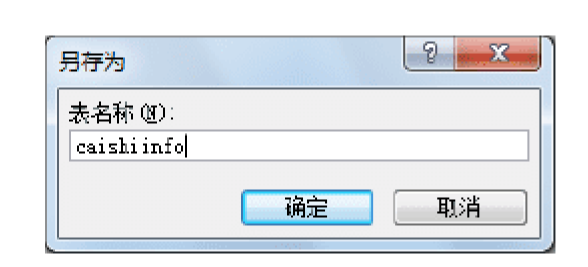


图 2.15　新建数据表





（2）创建字段名称及数据类型

单击图 2.15 所示的“确定”按钮，将弹出如图 2.16 所示的窗口。

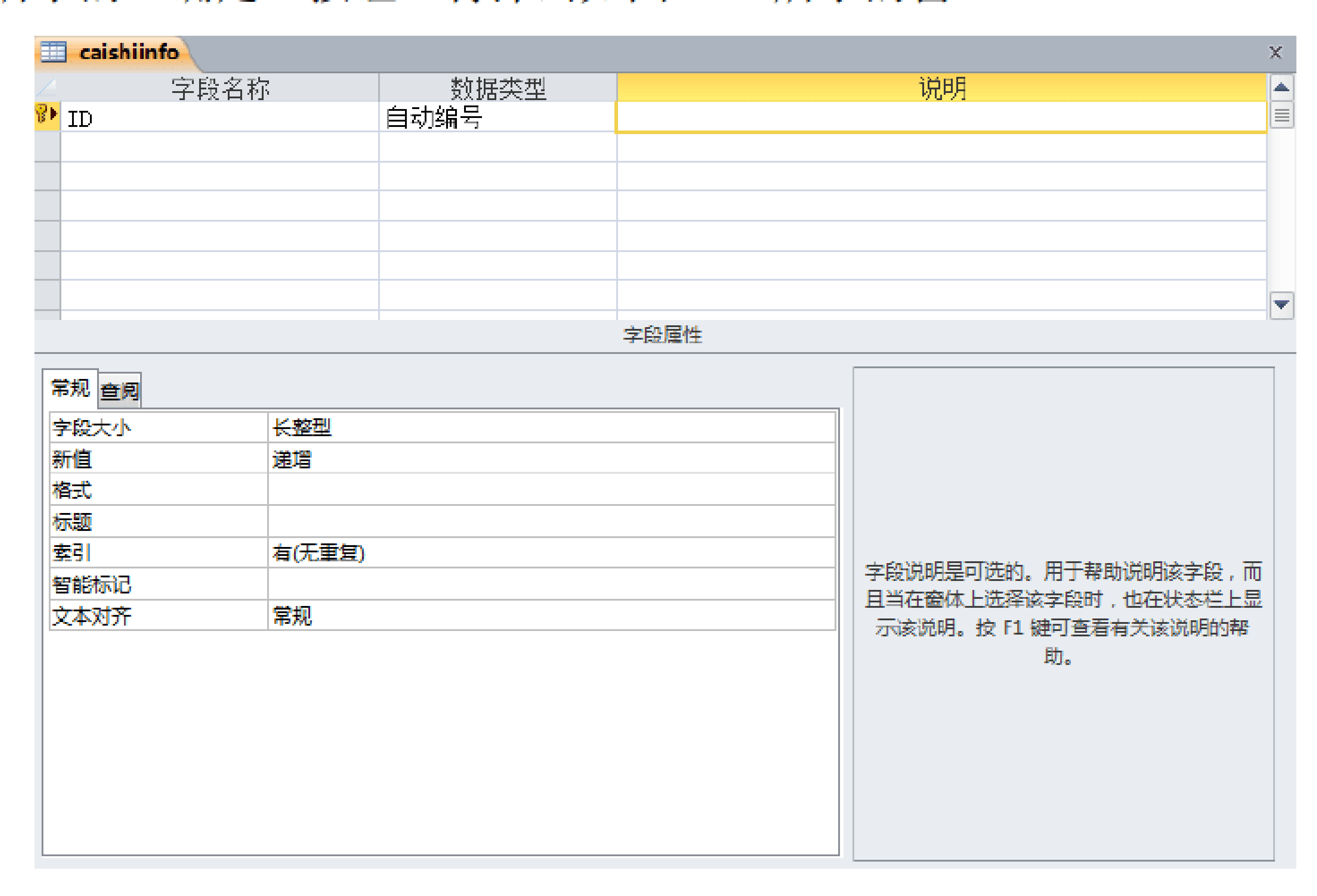


图 2.16　创建字段名称及其数据类型

（3）输入信息

在“字段名称”中分别输入菜名和菜价，再将数据类型分别设置为自动编号、文本和数字，如图 2.17 所示。

（4）保存表

设置完毕后，在菜单栏中选择“文件”→“保存”命令，将表格保存，完成表的创建。

其余表的创建方法基本一致，下面分别介绍餐饮管理系统中各数据表的结构。

☑　菜式信息表（caishiinfo）：主要用于记录菜式信息，包括菜式名称和菜式价格，如图 2.18 所示。

☑　进货信息表（jinhuoinfo）：主要用于记录进货信息，方便使用者查询，如图 2.19 所示。

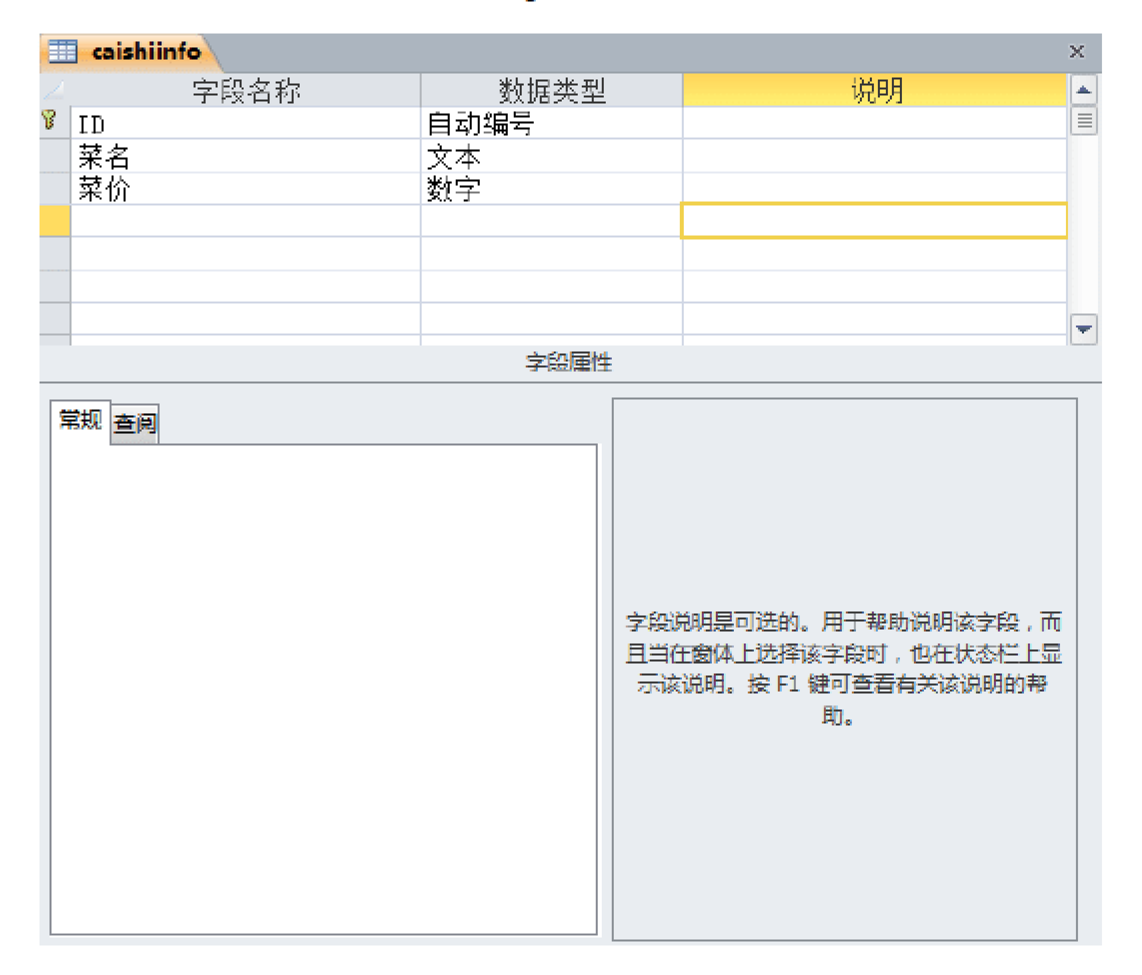


图 2.17　设置参数

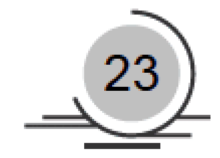


图 2.18　菜式信息表结构图

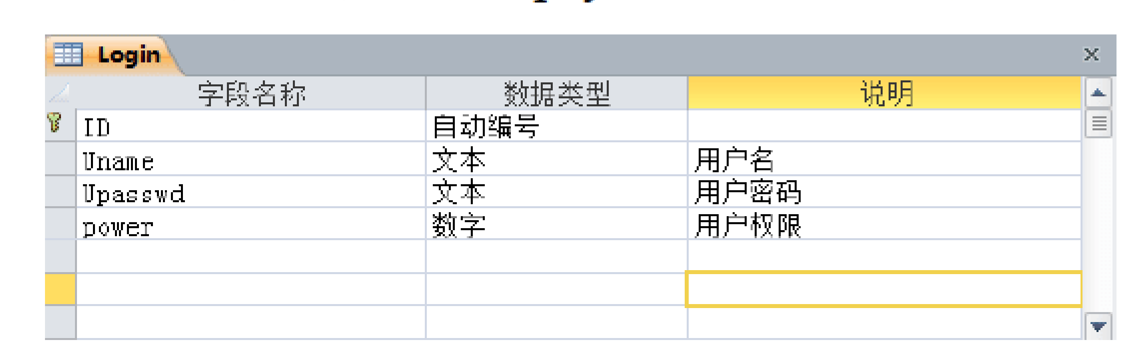
jinhuoinfo

字段名称	数据类型	说明
ID	自动编号	
进货时间	文本	进货的具体日期
商品名	文本	进货商品名称
商品数量	数字	进货商品数量
商品价格	数字	进货商品总花费

图 2.19　进货信息表结构图



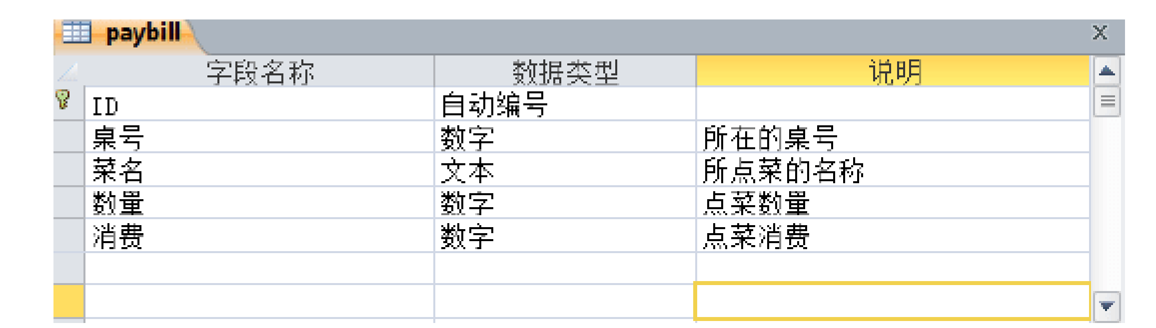
- ☑ 用户信息表（Login）：主要用于保存用户账号、密码和权限等信息，如图 2.20 所示。
- ☑ 账单信息表（paybill）：主要用于保存顾客的消费信息，如图 2.21 所示。



Login

字段名称	数据类型	说明
ID	自动编号	
Uname	文本	用户名
Upasswd	文本	用户密码
power	数字	用户权限

图 2.20　用户信息表结构图

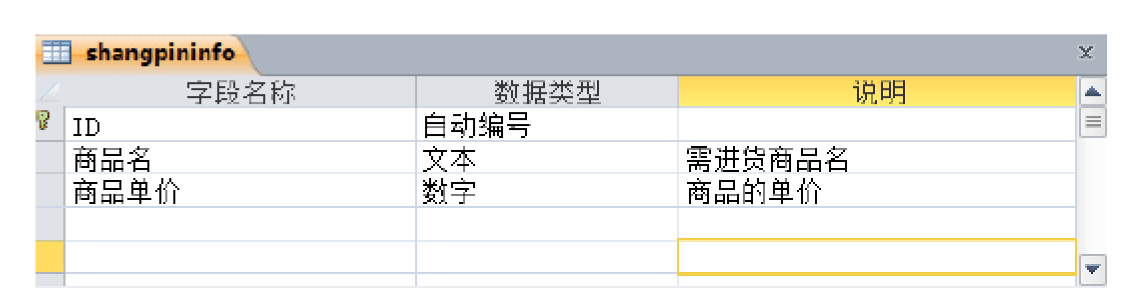


paybill

字段名称	数据类型	说明
ID	自动编号	
桌号	数字	所在的桌号
菜名	文本	所点菜的名称
数量	数字	点菜数量
消费	数字	点菜消费

图 2.21　账单信息表结构图

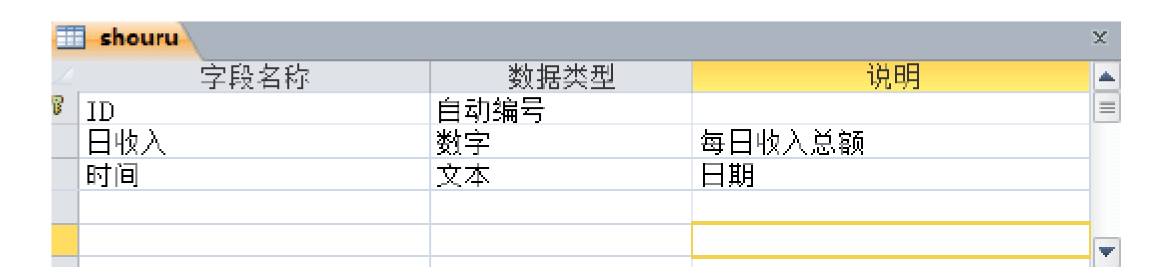
- ☑ 商品信息表（shangpininfo）：主要用于登记需要进货的商品信息，包括商品名称及价格，如图 2.22 所示。
- ☑ 收入信息表（shouru）：主要用于记录每天的总营业额信息，以方便用户查询日收入总额及月收入总额情况，如图 2.23 所示。



shangpininfo

字段名称	数据类型	说明
ID	自动编号	
商品名	文本	需进货商品名
商品单价	数字	商品的单价

图 2.22　商品信息表结构图

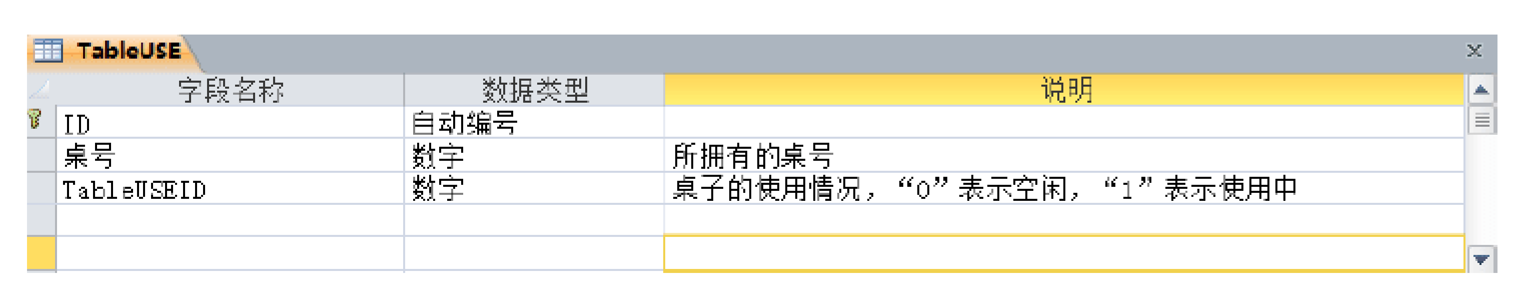


shouru

字段名称	数据类型	说明
ID	自动编号	
日收入	数字	每日收入总额
时间	文本	日期

图 2.23　收入信息表结构图

- ☑ 餐桌使用情况表（TableUSE）：主要用于记录每个餐桌的使用情况，如图 2.24 所示。



TableUSE

字段名称	数据类型	说明
ID	自动编号	
桌号	数字	所拥有的桌号
TableUSEID	数字	桌子的使用情况，“0”表示空闲，“1”表示使用中

图 2.24　餐桌使用情况表结构图



2.4　公共类设计

设计系统时，经常会重复使用同一种功能模块，为避免代码重复使用率过高，往往将重复使用频率高的代码写成公共类。

数据库连接是系统中必不可少的部分，在每个模块中都需要连接数据库进行数据操作。为此，笔者将数据库连接方法写在程序的 App 类中。

设计步骤如下：

（1）在工作区窗口选择 FileView 选项卡，在 Header Files 目录下找到头文件 StdAfx.h，向其中添加如下代码（路径根据实际情况更换），用于将 msado15.dll 动态链接库导入程序中，如图 2.25 所示。

```
#import "C:\\Program Files\\Common Files\\System\\ado\\msado15.dll"no_namespace rename("EOF","adoEOF")
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__9B887328_1300_4F88_B2A0_EBEFD1520770__INCLUDED_)
#define AFX_STDAFX_H__9B887328_1300_4F88_B2A0_EBEFD1520770__INCLUDED_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#import "C:\\Users\\zhangyan\\Desktop\\04\\Programe\\msado15.dll" no_namespace rename("EOF","adoEOF")

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__9B887328_1300_4F88_B2A0_EBEFD1520770__INCLUDED_)
```

在这导入

图 2.25　导入动态链接库

（2）在 App 类的 InitInstance 方法中添加代码，设置数据库连接，因为 App 类中有全局变量 TheApp，所以在 App 类中连接数据库后可以方便地使用全局变量对其进行操作，代码如下：

```
BOOL CMyApp::InitInstance()
{
    AfxEnableControlContainer();
    ::CoInitialize(NULL);
    HRESULT hr;                                              //定义一个 HRESULT 实例
    try
    {
❶       hr=m_pCon.CreateInstance("ADODB.Connection");        //创建连接
        if(SUCCEEDED(hr))                                    //判断创建连接是否成功
        {
            m_pCon->ConnectionTimeout=3;                     //连接延时设置为 3 秒
❷           hr=m_pCon->Open("Provider=Microsoft.Jet.OLEDB.4.0;Data
                        Source=canyin.mdb","","",adModeUnknown);   //连接数据库
        }
    }
    catch(_com_error e)
    {
        CString temp;
        temp.Format("连接数据库错误信息:%s",e.ErrorMessage());   //获得错误信息
        ::MessageBox(NULL,temp,"提示信息",NULL);                 //弹出错误信息
        return false;
    }
//以下代码省略
…
    return FALSE;
}
```

代码贴士

❶ 在调用 CreateInstance 函数时，使用的不是指针调用形式“→”，因为 m_pConnection 虽然是指针类型，但是 CreateInstance 函数不是指针所指向的对象方法，而是只能指针本身的函数，所以在调用时使用的是“.”的形式。

❷ 连接字符串通常是通过向程序中导入一个 ADO Data 控件，再通过设置 ADO Data 控件的连接属性获得的。

（3）代码添加完成后，各个模块就可以通过 App 类的全局变量 theApp 直接操作数据库了。



2.5　主窗体设计

程序主窗体作为第一个展示在用户面前的窗体，是用户对程序的第一感觉，在程序中起着非常重要的作用。主窗体应该向用户展示程序常用的功能，使用户对程序有一个初步的认识。主窗体的运行效果如图 2.26 所示。

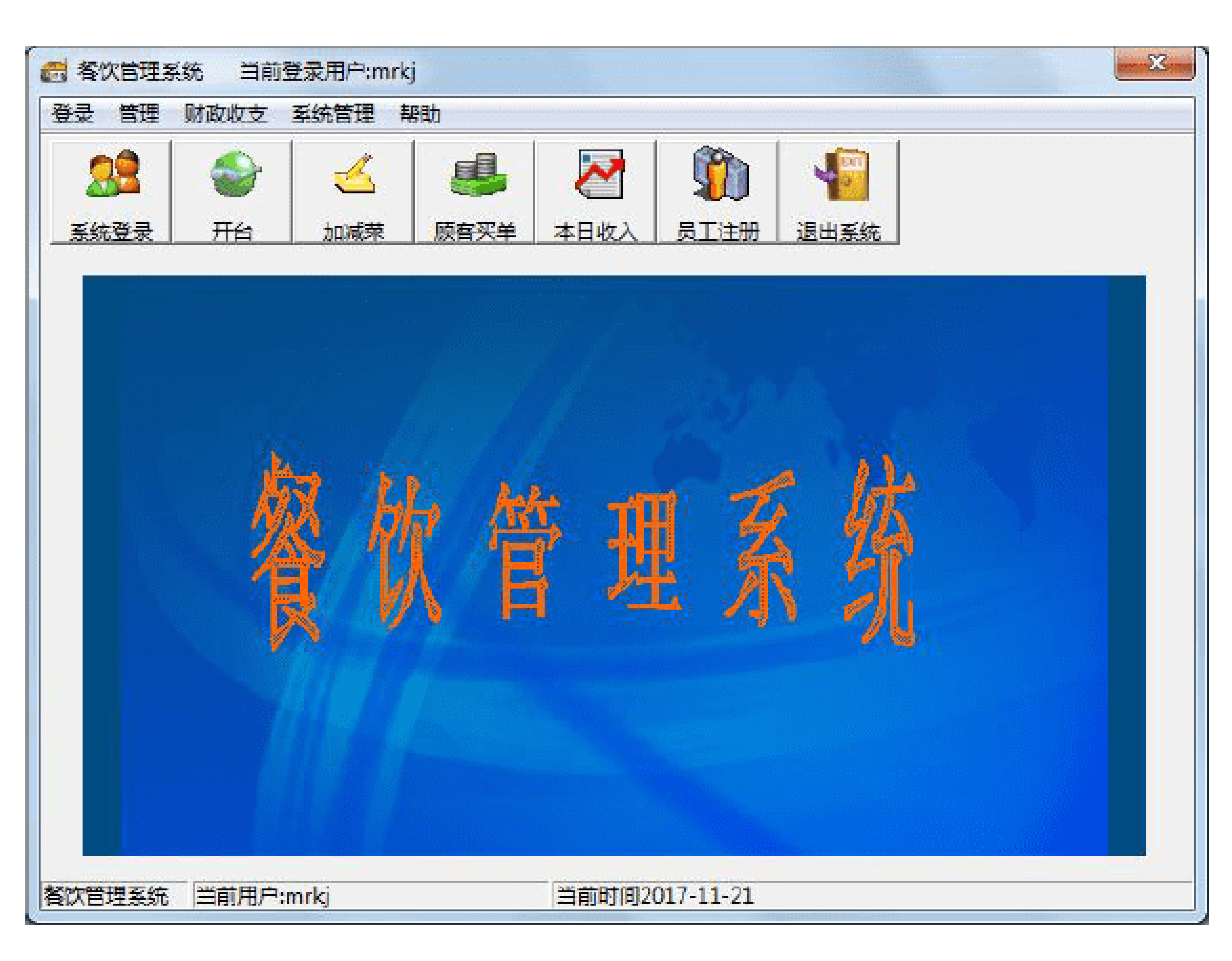


图 2.26　程序主窗体的运行效果

主窗体主要包含以下内容。

☑　菜单栏：包括登录、前台服务和后台服务等一系列程序所拥有的功能。

☑　工具栏：包括程序比较常用的几个功能，如开台、顾客买单等。

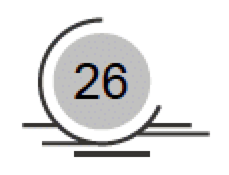
☑　状态栏：包括系统的名称、当前时间及用户登录信息等。

设计步骤如下：

（1）启动 Visual C++ 6.0，新建一个基于对话框的 MFC 应用程序，并将程序命名为“餐饮管理”，如图 2.27 所示。

（2）单击 OK 按钮后弹出如图 2.28 所示的对话框，选中 Dialog based 单选按钮，单击 Finish 按钮完成创建。

（3）单击 Finish 按钮后，在工作区中选择 Resources 选项卡，在任意一个节点上右击，在弹出的快捷菜单中选择 Insert 命令，打开 Insert Resource 对话框。在 Resource type 列表中选择 Menu 节点，单



击 New 按钮，将创建一个菜单，在菜单设计窗口中，按 Enter 键打开属性窗口，设计菜单标题，完成后在窗体 Menu 选项中修改生成的菜单 ID，如图 2.29 所示。

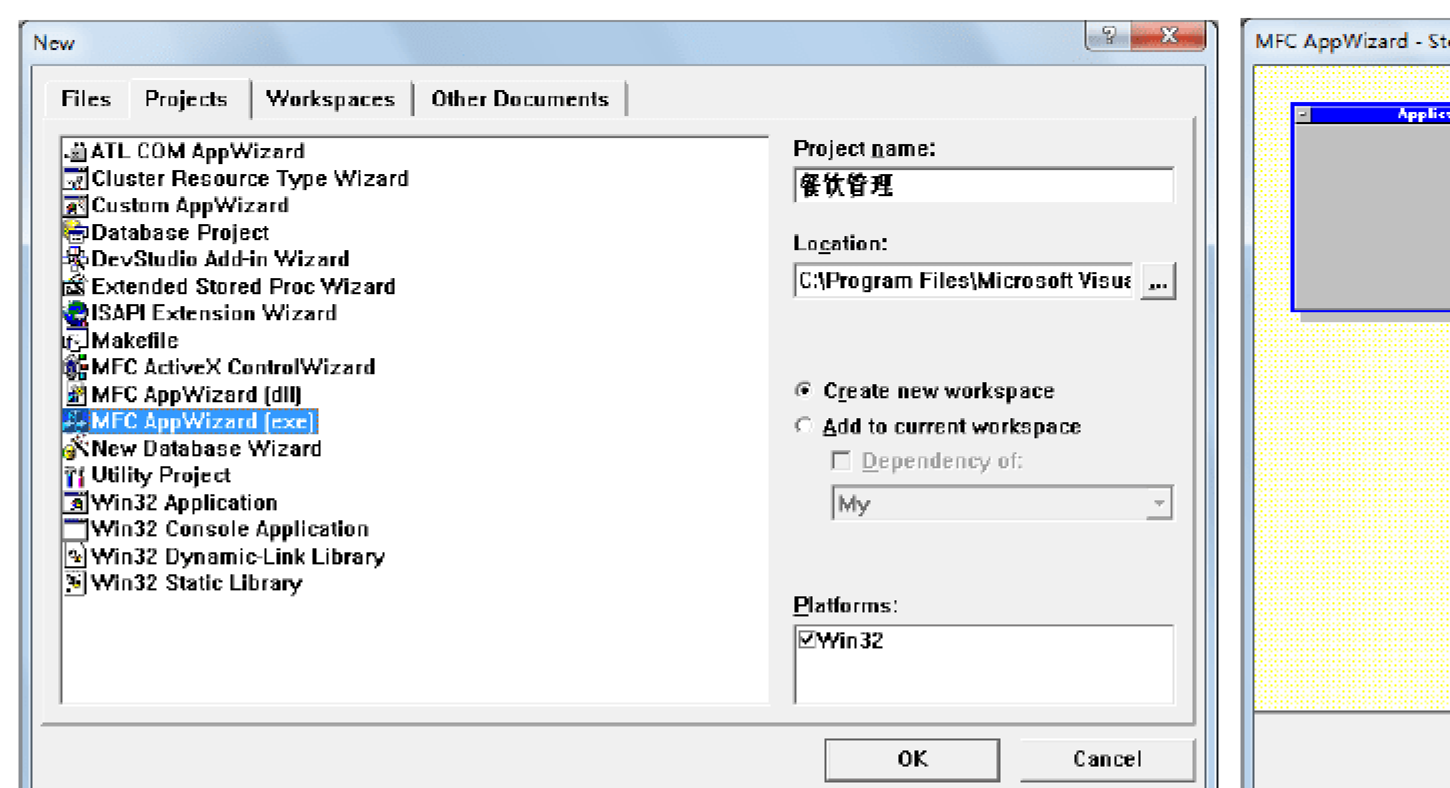


图 2.27　新建一个 MFC 程序

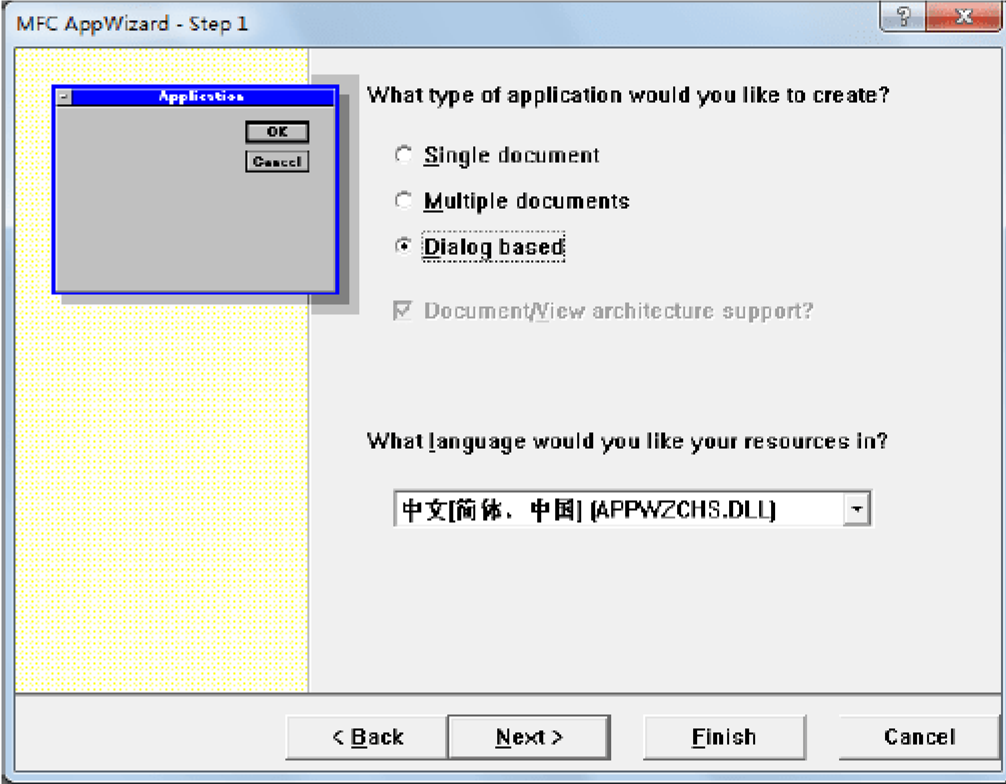


图 2.28　程序的创建

（4）由于生成的是带图标的工具栏，所以需要事先在 Resources 选项卡中选择 Insert 菜单项导入几个图标文件，如图 2.30 所示。



图 2.29　创建菜单项

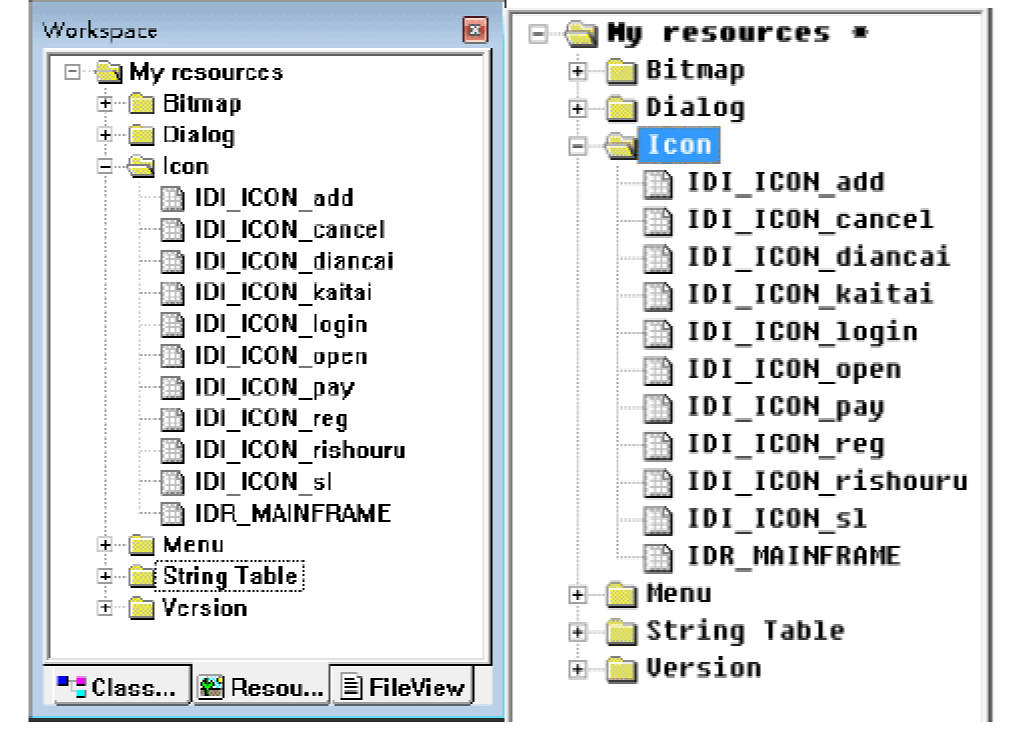


图 2.30　导入图标文件

（5）在生成的窗口类的 OnInitDialog 方法中添加代码动态生成工具栏和状态栏，代码如下：

```
m_Imagelist.Create(32,32,ILC_COLOR24|ILC_MASK,1,1);          //创建图像列表
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_login));      //将图像与列表一一关联
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_open));
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_pay));
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_rishouru));
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_reg));
m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_cancel));
UINT Array[6];                                                //数组控制工具栏和状态栏的个数
for(int i=0;i<6;i++)
{
    Array[i]=9000+i;                                          //分别给工具栏的按钮定义索引
```

```
}
m_Toolbar.Create(this);                                          //创建工具栏资源
m_Toolbar.SetButtons(Array,7);                                   //设置 6 个按钮
m_Toolbar.SetButtonText(0,"系统登录");                           //给每个按钮添加文本
m_Toolbar.SetButtonText(1,"开台");
m_Toolbar.SetButtonText(2,"加减菜");
m_Toolbar.SetButtonText(3,"顾客买单");
m_Toolbar.SetButtonText(4,"本日收入");
m_Toolbar.SetButtonText(5,"员工注册");
m_Toolbar.SetButtonText(6,"退出系统");
m_Toolbar.GetToolBarCtrl().SetButtonWidth(60,120);               //设置按钮宽度
m_Toolbar.GetToolBarCtrl().SetImageList(&m_Imagelist);           //将工具栏和图标关联
//设置按钮大小和图片大小
m_Toolbar.SetSizes(CSize(70,60),CSize(28,40));
m_Toolbar.EnableToolTips(TRUE);                                  //激活鼠标提示功能
for(i=0;i<4;i++)
{
    Array[i]=10000+1;                                            //分别给状态栏定义索引
}
m_Statusbar.Create(this);                                        //创建状态栏资源
m_Statusbar.SetIndicators(Array,4);                              //设置 4 个状态栏
for(int n=0;n<3;n++)
{
    m_Statusbar.SetPaneInfo(n,Array[n],0,80);                    //给每个状态栏设置宽度
}
❶ m_Statusbar.SetPaneInfo(1,Array[1],0,200);
m_Statusbar.SetPaneInfo(2,Array[2],0,800);
❷ m_Statusbar.SetPaneText(2,"当前时间"+Str);                     //设置状态栏的文本
m_Statusbar.SetPaneText(0,"餐饮管理系统");
//显示工具栏和状态栏
❸ RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0);
```

代码贴士

❶ SetPaneInfo：设置指定状态栏面板的宽度。

❷ SetPaneText：设置指定状态栏面板中显示的文本。

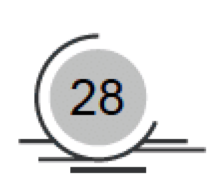
❸ RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,0)：用于显示工具栏和状态栏的函数，在工具栏和状态栏都存在的情况下只需输入一次即可。



2.6 注册模块设计

2.6.1 注册模块概述

注册模块是一个完善的管理系统中必不可少的部分，主要用于预防非法用户随意登录系统并对





系统数据进行修改破坏，给经营者造成不可挽回的损失。只有系统管理者才能通过注册模块对指定的人员进行注册，使其可以对系统进行相应的操作，大大提高了系统的安全性。注册模块的运行效果如图 2.31 所示。

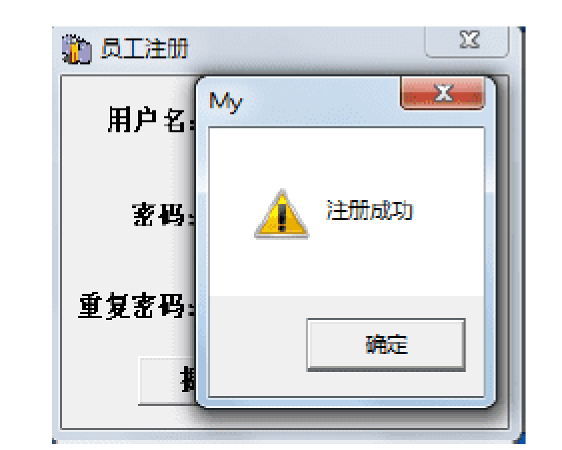


图 2.31　注册模块效果图

2.6.2　注册模块技术分析

在此模块中主要知识点是 SQL 语句的灵活运用，通过向数据表中直接添加数据即可达到实现用户注册的目的，添加数据可以用 INSERT 语句来实现。在此也介绍了 SQL 语句的执行方法 Execute，通过连接对象的 Execute 方法可以很容易地执行 INSERT 语句。

Execute 方法的语法如下：

```
Connection Execute(_bstr_t CommandText,VARIANT * RecordsAffected,long Options)
```

☑　CommandText：命令字符串，通常是 SQL 命令。

☑　RecordsAffected：操作后所影响的行数。

☑　Options：CommandText 中内容的类型，其值如表 2.1 所示。

表 2.1　Options 值表

值	描　　述
adCmdText	表明 CommandText 的类型是文本
adCmdTable	表明 CommandText 的类型是表名
adCmdStoredProc	表明 CommandText 的类型是存储过程
adCmdUnknown	表明 CommandText 的类型未知

INSERT 语句的基本语法如下：

```
INSERT INTO [表名](需要插入的列名) values(要插入的数值)
```

例如，读者想向用户注册信息表中插入一条用户信息，INSERT 语句可写为：

```
CString sql;
sql.Format("INSERT INTO register(username,userpasswd) values('%s', '%s')");
```

接下来要执行这条语句就可以使用 Execute 方法。

```
m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);   //m_pCon 是一个数据库连接对象
```

2.6.3　注册模块实现过程

本模块使用的数据表：Login

（1）在 Resources 选项卡中插入一个对话框资源，在对话框中添加 3 个静态文本控件、3 个编辑框控件和两个按钮控件。控件的属性及变量如表 2.2 所示。

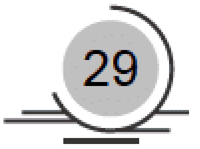


表 2.2　控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_STATIC	标题：用户名	无
IDC_STATIC	标题：密码	无
IDC_STATIC	标题：重复密码	无
IDC_EDIT_name	Visible	CString m_Name
IDC_EDIT_pwd	Password	CString m_Pwd
IDC_EDIT_pwd1	Password	CString m_Pwd1
IDC_BUTTON_OK	标题：提交	无
IDC_BUTTON_reset	标题：重置	无

（2）给对话框新建一个类 CZhucedlg，在类中添加一个_RecordsetPtr 类型变量 m_pRs 并导入全局变量 theApp。

（3）双击注册模块对话框中的“提交”按钮，在弹出的函数名称窗口中定义函数名称，单击“确定”按钮，进入按钮的代码编写界面。

当用户单击“提交”按钮时，系统应该判断输入的用户名是否跟数据表中的用户名重复，如果重复则弹出提示对话框；再判断两次密码输入是否一致，如果不一致则需弹出提示对话框要求重新输入，成功后则向数据表中插入用户名、密码和权限（默认权限为 0）信息，代码如下：

```
UpdateData();
//判断“用户名”和“密码”编辑框是否为空
if(m_Name.IsEmpty()||m_Pwd.IsEmpty()||m_Pwd1.IsEmpty())
{
    AfxMessageBox("用户名密码不能为空");
    return;
}
if(m_Pwd!=m_Pwd1)                                //判断两次输入的密码是否一致
{
    AfxMessageBox("密码不一致");
    return;
}
//检验数据表中用户名是否重复
m_pRs=theApp.m_pCon->Execute((_bstr_t)("select * from Login where
                                    Uname='"+m_Name+"'"),NULL,adCmdText);
if(m_pRs->adoEOF)                                //判断记录是否为空
{
    //如果为空，就向数据表中插入用户名、密码及权限信息
    theApp.m_pCon->Execute((_bstr_t)("insert into Login(Uname,Upasswd,power)
values('"+m_Name+"', '"+m_Pwd+"',0)"),NULL,adCmdText);
    AfxMessageBox("注册成功");
    CDialog::OnOK();
}
else                                             //如果不为空，就提示用户名重复
{
```



```
        AfxMessageBox("用户名已存在");
        return;
}
```

（4）为“重置”按钮添加代码，“重置”按钮主要实现的功能是把对话框中的 3 个编辑框控件的状态设置为初始状态，代码如下：

```
m_Name="";
m_Pwd="";
m_Pwd1="";
UpdateData(false);
```

2.7　登录模块设计



2.7.1　登录模块概述

在本系统中，登录模块的功能是判断用户是不是合法用户，以及根据登录用户的权限开放相应的模块，是保障系统安全的第一道关卡。登录模块的运行效果如图 2.32 所示。



图 2.32　登录模块的运行效果

2.7.2　登录模块技术分析

在登录模块中，为了避免个别人恶意猜测他人的用户名和密码，笔者在系统中添加了密码错误次数限制，如果密码输入错误次数超过 3 次，就会退出程序。

为了实现以上功能，需要在登录类中添加一个全局变量计算输入错误密码的次数，因为本系统登录时调用的是模块对话框，所以在关闭时就必须先关闭当前的登录模块，再关闭程序主界面。在登录类的 OK 按钮的代码中加入对次数的判断，如果次数等于 3 就调用本对话框的退出事件；再在主界面的“登录”按钮代码中对错误次数进行判断，如果次数等于 3 就调用主对话框的退出事件。要实现这一功能，需先在主对话框的“登录”按钮代码中加入如下代码：

```
if(Logindlg.i==3) CDialog::OnCancel();                //Logindlg 是登录模块的一个实例
```

判断登录模块中的 i 值是否为 3，如果 i 值为 3 则调用主窗体的退出事件。在调用前应该先关闭登录模块对话框，所以在登录模块对话框的“确定”按钮中加入如下代码：

```
if(i==3) OnCancel();                                  //当 i=3 时调用“退出”按钮事件
```

当 i=3 时调用登录模块对话框中的“退出”按钮事件关闭对话框，OnCancel 方法是登录对话框的“退出”按钮事件。

2.7.3 登录模块实现过程

本模块使用的数据表：Login

（1）在 Resources 选项卡中插入一个对话框资源，向对话框中添加两个静态文本控件、两个编辑框控件、两个按钮控件和一个图片控件，打开图片控件的属性窗口给其关联一幅图片。控件的属性及变量如表 2.3 所示。

表 2.3 控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_STATIC	标题：用户名	无
IDC_STATIC	标题：密码	无
IDC_STATIC	Bitmap	无
IDC_EDIT1	Visible	CString m_Uname
IDC_EDIT2	Password	CString m_Upasswd
IDOK	标题：登录	无
IDCANCEL	标题：退出	无

（2）为登录模块新建一个 CLogindlg 类，在类中定义一个_RecordsetPtr 类型变量 m_pRs，在窗口类中添加代码导入全局变量 theApp，如图 2.33 所示，代码如下：

```
extern CMyApp theApp;
```

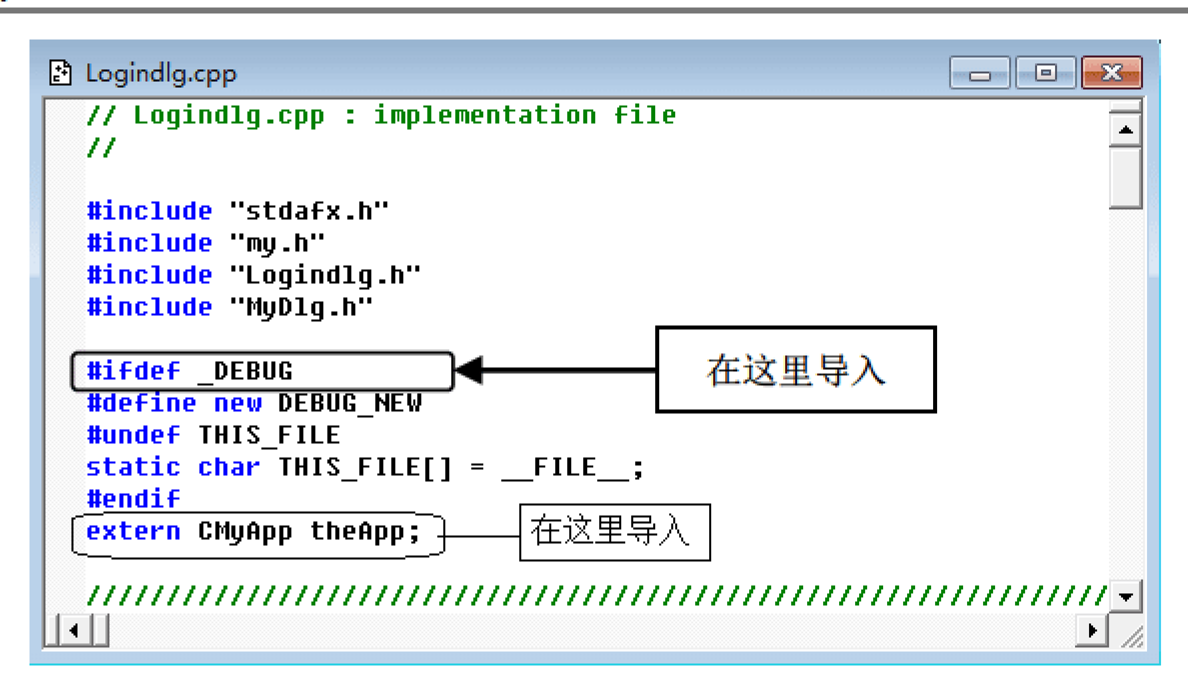


图 2.33 定义 theApp

（3）为“登录”按钮的单击事件添加代码，在“登录”按钮的单击事件下，系统应自动将用户输入的数据与数据表中的数据进行比较，如果都一致则提示成功登录；如果不一致则提示用户名、密码错误，代码如下：

```
UpdateData();
//判断“用户名”和“密码”编辑框是否为空
if(!m_Uname.IsEmpty()&&!m_Upasswd.IsEmpty()||true)
{
    CString sql="SELECT * FROM Login WHERE Uname='"+m_Uname+"' and Upasswd='"+m_Upasswd+"'";
```





```
		//在数据表中查询是否存在该用户名及密码
		m_pRs=theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
		if(m_pRs->adoEOF)					//如果没有账号记录则提示错误
		{
			AfxMessageBox("用户名或密码错误!");
			m_Uname="";
			m_Upasswd="";
			UpdateData(false);
			if(i==3)				//定义全局变量 i 控制输入错误次数
			{
				OnCancel();			//如果为 3 则调用退出事件
			}
		}
		else
		{
			theApp.name=m_Uname;		//登录成功后保存用户名和密码
			theApp.pwd=m_Upasswd;
			CDialog::OnOK();
			return;
		}
	}
	else						//如果编辑框为空则提示不能为空
	{
		AfxMessageBox("用户名密码不能为空");
	}
```



2.8　开台模块设计

2.8.1　开台模块概述

开台是餐饮系统中前台的第一个服务，顾客前来就餐时，卖家第一步应做的就是开台，开台模块应该直观地为用户展示当前空桌的情况，提高用户工作效率。开台模块的运行效果如图 2.34 所示。

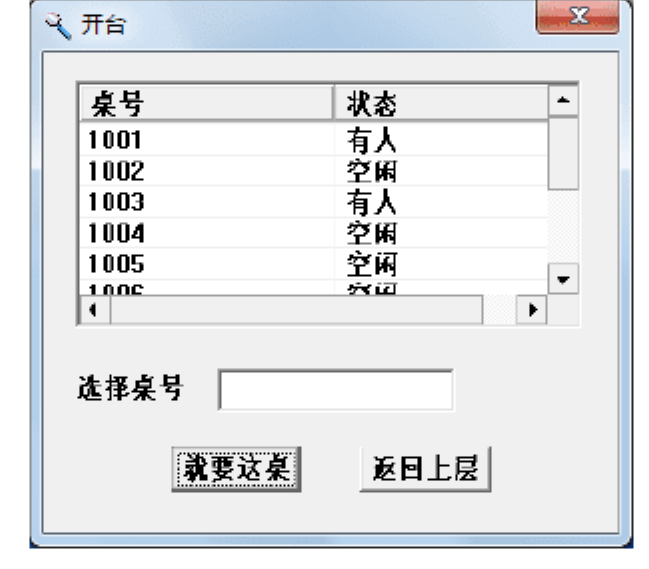


图 2.34　开台模块的运行效果

2.8.2　开台模块技术分析

在开台模块中，主要涉及对列表控件的使用，以及如何将数据表中的数据导入到列表控件中。在营业员为顾客进行选桌服务时，在餐桌使用情况信息表中双击要开台的桌台，即可将此桌台的桌号信息添加到“选择桌号”文本框中，大大地方便了使用者。要实现此功能，首先要在消息对话框左边的控件名称中找到列表控件，再在右边的事件中选择 NM_DBLCLK 事件，并为其添加相应的代码。在获取数据前，系统要先获取用户双击选项的

位置信息，可通过 GetSelectionMark 方法实现，再通过 GetItemText 方法获取当前位置的文本。这两个方法的语法如下：

```
int GetSelectionMark();
```

返回的是位置所在的行号，-1 表示没有位置。

```
CString GetItemText(int nItem, int nSubItem)
```

☑ nItem：表示位置所在行号。

☑ nSubItem：表示列号。

2.8.3　开台模块实现过程

本模块使用的数据表：TableUse

（1）在 Resources 选项卡中插入一个对话框资源，为对话框新建一个类 CKaitaidlg，向对话框中添加一个静态文本控件、一个列表控件和一个编辑框控件，在类中定义一个_RecordsetPtr 类型变量 m_pRs，并导入全局变量 theApp。控件的属性及变量设置如表 2.4 所示。

表 2.4　控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_STATIC	标题：选择桌号	无
IDC_LIST1	Report	CListCtrl m_Zhuolist
IDC_EDIT1	Visible	CString m_ZhuoHao
IDC_BUTTON_OK	标题：就要这桌	无
IDC_BUTTON_return	标题：返回上层	无

（2）为类添加 WM_INITDIALOG 事件并添加代码，进行对话框初始化设置并对列表控件的样式及内容进行设置，代码如下：

```
BOOL CKaitaidlg::OnInitDialog()
{
	CDialog::OnInitDialog();
//设置窗口图标
❶	SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDI_ICON_kaitai)),TRUE);
	//为列表控件设置样式
❷	m_Zhuolist.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_HEADER
DRAGDROP|LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
	//为列表控件添加两列并命名
	m_Zhuolist.InsertColumn(0,"桌号",LVCFMT_LEFT,140,0);
	m_Zhuolist.InsertColumn(1,"状态",LVCFMT_LEFT,140,1);
	CString sql="select * from tableuse";
	//查询数据表中的餐台号信息
	m_pRs=theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
	int i=0;				//控制列表控件中的显示顺序
	while(m_pRs->adoEOF==0)			//如果记录不为空，则遍历数据表并将结果添加进列表控件中
```

```
	{
	//将餐台号信息存入 str 变量
	CString str=(char*)(_bstr_t)m_pRs->GetCollect("桌号");
		int tableuseid=atoi((char*)(_bstr_t)m_pRs->GetCollect("tableuseid"));//将使用信息存入 tableuseid 变量
❸		m_Zhuolist.InsertItem(i,"");			//在列表控件中插入一行
	m_Zhuolist.SetItemText(i,0,str);			//将餐台号信息添加进该行第一列
	if(tableuseid==0)				//判断使用信息是否为 0
		m_Zhuolist.SetItemText(i,1,"空闲");	//如果为 0 就在该行第二列插入“空闲”
	if(tableuseid==1)				//判断使用信息是否为 1
		m_Zhuolist.SetItemText(i,1,"有人");	//如果为 1 就在该行第二列插入“有人”
	i++;						//控制行的变量自增
	m_pRs->MoveNext();				//移向下一条记录
	}
	return TRUE;
}
```

代码贴士

❶ SetIcon：该方法用于设置窗口图标，TRUE 是大图标，FALSE 是小图标。

❷ SetExtendedStyle：该方法可以为列表控件设置需要的风格。

❸ InsertItem：向列表控件中插入行。

（3）选择餐台号时不仅可以手动输入，而且要实现双击列表控件中的餐台号能直接将餐台号读进编辑框控件中。在消息管理器中选择列表控件的双击事件（NM_DBLCLK），添加函数并对其添加代码：

```
void CKaitaidlg::OnDblclkList1(NMHDR* pNMHDR, LRESULT* pResult)
{
	CString str;
	//获取当前列表控件中的鼠标单击位置所在行的第一列的文本
	str=m_Zhuolist.GetItemText(m_Zhuolist.GetSelectionMark(),0);
	m_ZhuoHao=str;				//将文本添加进编辑框中
	UpdateData(false);
	*pResult = 0;
}
```

运行后双击列表控件中的餐台号，系统自动将该桌台号的信息显示在下面的编辑框控件中。

（4）完成了界面效果的编辑，下一步对按钮控件进行编码。用户在单击“就要这桌”按钮时，系统应该先判断编辑框中输入的数据是否合法，如果不合法，则弹出输入错误的提示；如果合法，则弹出输入成功的提示，并进入“点菜”对话框。

说明 模块中涉及弹出“点菜”对话框的功能，故类中应包含点菜模块（CDiancaidlg）的头文件。点菜模块将在 2.9 节介绍，希望读者将这两节联系到一起阅读，方便理解。

“就要这桌”按钮的单击事件代码如下：

```
UpdateData();
CString Value;
```

```
if(m_ZhuoHao.IsEmpty())                              //判断编辑框是否为空
    AfxMessageBox("桌号不能为空");                   //如果为空则提示不能为空
else
{
    //如果不为空则查询哪些餐台正在使用
    CString Str="select * from TableUSE where TableUSEID=1";
    m_pRs=theApp.m_pCon->Execute((_bstr_t)Str,NULL,adCmdText);
    while(!m_pRs->adoEOF)                            //当记录不为空时
    {
        //将正在使用的餐台号存进变量中
        Value=(char*)(_bstr_t)m_pRs->GetCollect("桌号");
        if(m_ZhuoHao==Value)                         //将编辑框的值与变量相比较
        {
            AfxMessageBox("有人了");                 //如果相等则提示“有人了”
            m_ZhuoHao="";                            //编辑框初始化显示
            UpdateData(false);
            return;
        }
        m_pRs->MoveNext();                           //继续下一条记录
    }
    m_pRs=NULL;                                      //指针位置初始化
    //餐台没被使用时再查询是否存在这个餐台号
    CString Str1="select * from TableUSE where  桌号="+m_ZhuoHao+"";
    m_pRs=theApp.m_pCon->Execute((_bstr_t)Str1,NULL,adCmdText);
    if(m_pRs->adoEOF)                                //如果记录为空
    {
        AfxMessageBox("没有这种餐台");               //则提示不存在这样的餐台
        m_ZhuoHao="";                                //编辑框初始化显示
        UpdateData(false);
        return;
    }
    m_pRs=NULL;                                      //记录集指针初始化
    CDiancaidlg dlg;                                 //定义一个点菜窗体实例
    //将编辑框控件中的数据传递给点菜窗体中的变量
    dlg.m_ZhuoHao = m_ZhuoHao;
    dlg.DoModal();                                   //弹出点菜窗体
    CDialog::OnOK();
}
```

“返回上层”按钮的单击事件其实就是关闭当前对话框，代码如下：

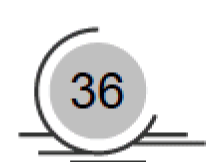
```
CDialog::OnCancel();
```



2.9 点菜模块设计

2.9.1 点菜模块概述

点菜模块和开台模块密不可分，在为顾客开台后会自动弹出“点菜”对话框为顾客点菜。点菜模





块运行效果如图 2.35 所示。

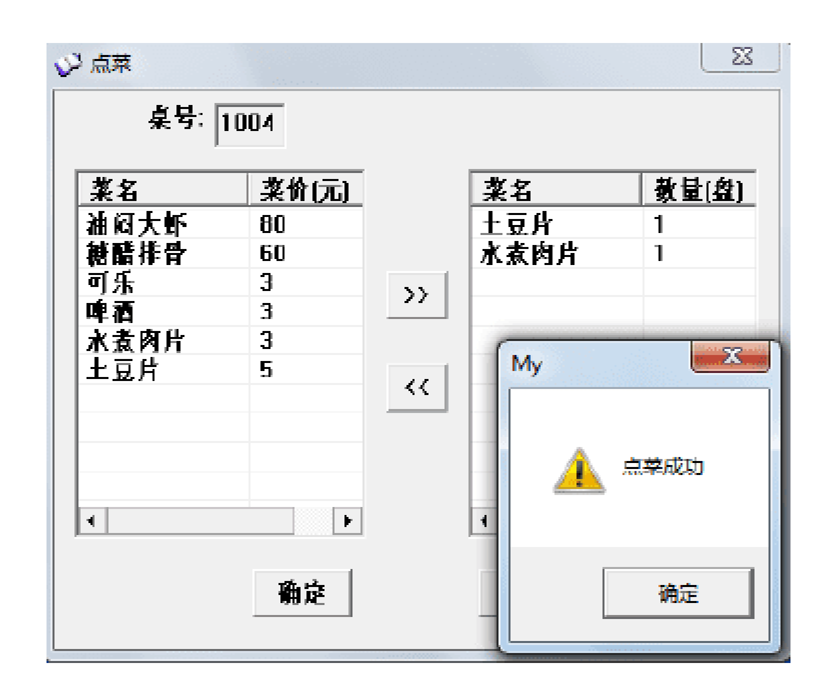


图 2.35　点菜模块运行效果

2.9.2　点菜模块技术分析

在点菜模块中主要应用了两个列表控件之间的数据传递技术，即从菜单中选择顾客所需要的菜式并将其添加到顾客的账单列表中。在传递的过程中，菜单列表是不能被修改的，账单列表要在每加进一样菜式时就必须增加一行数据，而在逆向传递时，账单列表的数据要相应减少，但菜单列表中数据不变。菜单列表应该采取直接从数据库中读取的方式，以防遭人恶意修改，在单击“确定”按钮前，所有的数据应该都只在列表控件中进行传递而不写入数据库，从而保证数据库的安全性。在获取列表控件当前鼠标指针所在位置时可以用 2.8 节提到的 GetSelectionMark 方法得到。向列表中插入数据可以使用 SetItemText 方法，该方法用于设置视图项的文本，语法如下：

```
BOOL SetItemText(int nItem, int nSubItem, LPTSTR lpszText);
```

☑　nItem：标识行索引。
☑　nSubItem：标识列索引。
☑　lpszText：标识设置的视图项文本。

2.9.3　点菜模块实现过程

本模块使用的数据表：TableUSE、caishiinfo、paybill

1．顾客点菜

（1）在 Resources 选项卡中插入一个对话框资源，为对话框新建一个类 CDiancaidlg，在类中定义一个_RecordsetPtr 类型变量 m_pRs 并导入全局变量 theApp。在对话框中添加两个列表控件、一个静态文本控件、一个编辑框控件和 4 个按钮控件。控件的属性及变量如表 2.5 所示。

表 2.5　控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_STATIC	标题：桌号	无
IDC_LIST2	Report	CListCtrl m_CaidanList
IDC_LIST3	Report	CListCtrl m_CaidanCheck
IDC_EDIT_zhuohao	Read-Only	CString m_ZhuoHao

（2）为 CDiancaidlg 类添加一个 WM_INITDIALOG 消息，用于设置列表控件的样式及内容，代码如下：

```
BOOL CDiancaidlg::OnInitDialog()
{
	CDialog::OnInitDialog();
```



```
    //为窗体设置图标
    SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDI_ICON_diancai)),TRUE);
    CString Sql="select * from caishiinfo";                    //查询菜式信息
    //为菜单列表进行样式设置
    m_CaidanList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|
            VS_EX_HEADERDRAGDROP|LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
❶   m_CaidanList.InsertColumn(0,"菜名",LVCFMT_LEFT,100,0);   //为菜单列表添加两列并命名
    m_CaidanList.InsertColumn(1,"菜价(元)",LVCFMT_LEFT,100,1);
    //读取数据表中菜单的信息向列表控件中添加
    m_pRs=theApp.m_pCon->Execute((_bstr_t)Sql,NULL,adCmdText);
    while(!m_pRs->adoEOF)                                      //当记录集指针不为空时
    {
        CString TheValue,TheValue1;
        //将菜名信息存入变量 TheValue
        TheValue=(char*)(_bstr_t)m_pRs->GetCollect("菜名");
        //将菜价信息存入变量 TheValue1
        TheValue1=(char*)(_bstr_t)m_pRs->GetCollect("菜价");
        m_CaidanList.InsertItem(0,"");                         //为列表框插入一行
❷       m_CaidanList.SetItemText(0,0,TheValue);                //将该行的第一列设置为文本
        m_CaidanList.SetItemText(0,1,TheValue1);               //将该行的第二列设置为文本
        m_pRs->MoveNext();                                     //继续下一条记录
    }
    //为菜单选择列表进行样式设置
    m_CaidanCheck.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_HEADERD
RAGD

ROP|LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
    //为菜单选择列表添加两列并命名
    m_CaidanCheck.InsertColumn(0,"菜名",LVCFMT_LEFT,100,0);
    m_CaidanCheck.InsertColumn(1,"数量(盘)",LVCFMT_LEFT,100,1);
    return TRUE;
}
```

代码贴士

❶ InsertColumn：该方法主要有 LVCFMT_LEFT（向左靠齐）、LVCFMT_RIGHT（向右靠齐）和 LVCFMT_CENTER（居中靠齐）3 种。

❷ SetItemText：该方法用于向列表中指定行、指定列并插入数据。

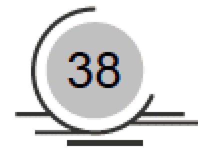
在“点菜”对话框中，编辑框控件的值来自开台模块的“就要这桌”按钮，当开台确认后系统会自动将台号的值赋给编辑框控件，方便在数据表中进行数据存储。

（3）添加一个用于输入点菜数量的对话框，新建一个 CSLdlg 类，在对话框中添加一个静态控件、一个编辑框控件和两个按钮控件，如图 2.36 所示。

（4）为“点菜数量”对话框的编辑框控件添加一个 CString 型变量 m_ShuLiang。先给对话框添加一个对话框图标，要想实现这一功能，先要在 Resources 选项卡中插入一个图标资源，再为 CSLdlg 类添加一个 WM_INITDIALOG 消息，向其添加如下代码：

图 2.36　“点菜数量”对话框

```
SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDI_ICON_sl)),TRUE);
```





（5）为“点菜数量”对话框中的“确定”按钮添加代码，当用户单击“确定”按钮时系统将判断用户是否输入数据，数量至少要为 1，代码如下：

```
UpdateData();
if(m_ShuLiang.IsEmpty()||m_ShuLiang=="0")                //判断数量编辑框是否为空或是否为 0
{
    AfxMessageBox("数量至少为 1");                        //如果是则提示至少为 1
    return;
}
CDialog::OnOK();
```

（6）为“点菜数量”对话框中的“返回”按钮添加代码，当用户单击“返回”按钮时系统将进入点菜窗体，代码如下：

```
CDialog::OnCancel();
```

（7）“点菜”对话框中的“>>”按钮用于将菜单中的菜式名称添加进顾客的点菜列表中，代码如下：

```
void CDiancaidlg::OnButtonadd()
{
    CSLdlg Sldlg;
    if(Sldlg.DoModal()==IDOK)                            //单击“>>”按钮前要求输入数量
    {
        int i = m_CaidanList.GetSelectionMark();          //获取菜单中所选择的项的序号
        CString str = m_CaidanList.GetItemText(i,0);      //获取选择项的文本
        m_CaidanCheck.InsertItem(0,"");
        m_CaidanCheck.SetItemText(0,0,str);               //将文本写进点菜栏中
        //将数量写进点菜栏中
        m_CaidanCheck.SetItemText(0,1,Sldlg.m_ShuLiang);
    }
}
```

（8）“点菜”对话框中的“<<”按钮用于取消顾客已点的菜式，代码如下：

```
void CDiancaidlg::OnBUTTONsub()
{
    //删除点菜栏中所选择的项
    m_CaidanCheck.DeleteItem(m_CaidanCheck.GetSelectionMark());
}
```

（9）用户单击“确定”按钮时，系统将自动生成的账单添加进数据表中，代码如下：

```
void CDiancaidlg::OnButtonOk()
{
    UpdateData();
    CString Sql;
❶   int i = m_CaidanCheck.GetItemCount();               //获取点菜列表中项的总数
    if(i==0)                                            //如果项数为 0，则弹出提示
    {
        AfxMessageBox("请点菜");
```

```
            return;
        }
        Sql="update TableUSE set TableUSEID=1 where 桌号="+m_ZhuoHao+" ";
        //点菜成功则改变该餐台号的使用状态
        theApp.m_pCon->Execute((_bstr_t)Sql,NULL,adCmdText);
        CString Sql1,Str,Str1,Value,TotleValue;
        double Totle=0;                                    //定义一个变量存放营业额
        for(int n=0;n<i;n++)                               //遍历点菜列表，将数据存入数据表
        {
❷           Str=m_CaidanCheck.GetItemText(n,0);            //获取第 n 行第一列的数据信息
            Str1=m_CaidanCheck.GetItemText(n,1);           //获取第 n 行第二列的数据信息
            Sql1="select * from caishiinfo where 菜名='"+Str+"'";
            //获取菜价信息
            m_pRs=theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);
            Value=(char*)(_bstr_t)m_pRs->GetCollect("菜价");
            //将所选菜式的菜价与数量相乘算出总价
❸           Totle=atof(Value)*atof(Str1);
            TotleValue=(char*)(_bstr_t)Totle;
            //将此桌的点菜信息和消费明细写入数据表
            Sql1="insert into paybill(桌号,菜名,数量,消费)
                    values("+m_ZhuoHao+",'"+Str+"',"+Str1+","+TotleValue+")";
            theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);
        }
        AfxMessageBox("点菜成功");
        CDialog::OnOK();
}
```

代码贴士

❶ GetItemCount：获得列表控件中的节点数。

❷ GetItemText：获得列表控件指定行、指定列的数据。

❸ atof：将字符型数据转换为浮点型数据。

（10）单击“取消”按钮将关闭“点菜”对话框，代码如下：

```
CDialog::OnCancel();
```

2. 加菜减菜

顾客有时会要求餐厅加菜或减菜，本系统针对此类问题设置了加菜减菜模块，方便餐饮管理者更好地满足顾客的需求，如图 2.37 所示。



图 2.37 “加减菜”对话框

（1）在 Resources 选项卡中插入一个对话框资源，新建一个 CJiacaidlg 类，在类中定义一个_RecordsetPtr 类型变量 m_pRs 并导入全局变量 theApp。对其添加一个静态文本控件、一个下拉列表框控件、两个列表控件和 4 个按钮控件。控件的属性及变量如表 2.6 所示。

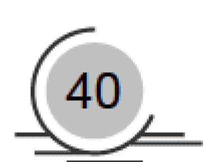




表 2.6　控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_COMBO1	Drop List	CComboBox m_ZhuohaoCombo
IDC_LIST2	Report	CListCtrl m_CaidanList
IDC_LIST3	Report	CListCtrl m_CaidanCheck

（2）先要对对话框的初始化进行设计，对列表控件的样式和内容进行初始化设置，对类添加消息函数 WM_INITDIALOG，代码如下：

```
BOOL CJiacaidlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	//为窗体设置图标
	SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDI_ICON_diancai)),TRUE);
	CString Sql="select * from caishiinfo";			//查询菜式信息表中的数据
	//对菜单列表进行样式设置
	m_CaidanList.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_HEAD
				ERDRAGDROP|LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
	//为菜单列表添加两列并分别命名
	m_CaidanList.InsertColumn(0,"菜名",LVCFMT_LEFT,100,0);
	m_CaidanList.InsertColumn(1,"菜价(元)",LVCFMT_LEFT,100,1);
	//将数据表中的菜单信息读入菜单列表中
	m_pRs=theApp.m_pCon->Execute((_bstr_t)Sql,NULL,adCmdText);
	while(!m_pRs->adoEOF)					//判断记录集指针是否为空
	{
		CString TheValue,TheValue1;
		//不为空则将菜名信息存入变量
		TheValue=(char*)(_bstr_t)m_pRs->GetCollect("菜名");
		//将菜价信息存入变量
		TheValue1=(char*)(_bstr_t)m_pRs->GetCollect("菜价");
		m_CaidanList.InsertItem(0,"");			//为菜单列表插入一行
		m_CaidanList.SetItemText(0,0,TheValue);		//将菜名信息添加进该行第一列
		m_CaidanList.SetItemText(0,1,TheValue1);	//将菜价信息添加进该行第二列
		m_pRs->MoveNext();				//继续下一条记录
	}
	//为点菜列表进行样式设置
	m_CaidanCheck.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_HEADERD
RAGD ROP|LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
	//为点菜列表添加两列并分别命名
	m_CaidanCheck.InsertColumn(0,"菜名",LVCFMT_LEFT,100,0);
	m_CaidanCheck.InsertColumn(1,"数量(盘)",LVCFMT_LEFT,100,1);
	Sql="select distinct 桌号 from paybill";		//去除重复的餐台号信息
	//向下拉列表框控件中添加数据
	m_pRs=theApp.m_pCon->Execute((_bstr_t)Sql,NULL,adCmdText);
	while(m_pRs->adoEOF==0)					//判断是否为空
	{
		//将餐台号信息存入变量
```

```
        CString zhuohao=(char*)(_bstr_t)m_pRs->GetCollect("桌号");
        m_ZhuohaoCombo.AddString(zhuohao);                    //为下拉列表框添加餐台号信息
        m_pRs->MoveNext();                                    //继续下一条记录
    }
    return TRUE;
}
```

（3）当下拉列表框控件的选项变化时，所选餐台号的菜单信息也应该相应改变，在消息对话框中下拉列表框控件的 SELCHANGE 事件代码如下：

```
void CJiacaidlg::OnSelchangeCombo1()
{
    CString str;
    //先获取所选选项的信息
    m_ZhuohaoCombo.GetLBText(m_ZhuohaoCombo.GetCurSel(),str);
    CString sql="select * from paybill where  桌号="+str+"";
    //到数据表中查找相关餐台号的数据信息
    m_pRs=theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
    m_CaidanCheck.DeleteAllItems();                           //菜单选择列表框初始化清空
    //将查找到的信息写入点菜列表中
    while(!m_pRs->adoEOF)                                     //判断记录集是否为空
    {
        //将菜名信息存入变量
        CString valuename=(char*)(_bstr_t)m_pRs->GetCollect("菜名");
        CString valuenum=(char*)(_bstr_t)m_pRs->GetCollect("数量");  //将数量信息存入变量
        m_CaidanCheck.InsertItem(0,"");                       //为菜单列表插入一行
        m_CaidanCheck.SetItemText(0,0,valuename);             //将菜名添加进该行第一列
        m_CaidanCheck.SetItemText(0,1,valuenum);              //将数量添加进该行第二列
        m_pRs->MoveNext();                                    //下一条记录
    }
}
```

（4）单击“>>”按钮将菜单中的菜式名称添加进用户当前账单中，代码如下：

```
void CJiacaidlg::OnButtonadd()
{
    CSLdlg Sldlg;
    if(Sldlg.DoModal()==IDOK)                                 //点菜前先添加数量信息
    {
        int i = m_CaidanList.GetSelectionMark();              //获取当前选中项的序号
        CString str = m_CaidanList.GetItemText(i,0);
        m_CaidanCheck.InsertItem(0,"");
        //将选中项的信息添加进点菜列表中
        m_CaidanCheck.SetItemText(0,0,str);
        //将数量信息添加到点菜列表
        m_CaidanCheck.SetItemText(0,1,Sldlg.m_ShuLiang);
    }
}
```



（5）单击“<<”按钮将从账单中取消用户刚刚所点的菜式名称，代码如下：

```
void CJiacaidlg::OnButtonsub()
{
	//删除点菜列表中所选的项
	m_CaidanCheck.DeleteItem(m_CaidanCheck.GetSelectionMark());
}
```

（6）单击“确定”按钮，系统将把已经变动的账单信息重新添加进数据表中，并将原始的账单信息删除，代码如下：

```
void CJiacaidlg::OnButtonOK()
{
	UpdateData();
	CString Sql;
	CString zhuohao;
	if(m_ZhuohaoCombo.GetCurSel()==-1)		//如果下拉列表框控件中没有选择数据则要求选择
	{
		AfxMessageBox("请选择要加菜的桌号");
		return;
	}
	//获取下拉列表框控件中所选择的信息
	m_ZhuohaoCombo.GetLBText(m_ZhuohaoCombo.GetCurSel(),zhuohao);
	int i = m_CaidanCheck.GetItemCount();			//获取点菜列表的项目总数
	if(i==0)							//如果点菜列表总数为 0，则提示请点菜
	{
		AfxMessageBox("请点菜");
		return;
	}
	CString Str,Str1,Value,TotleValue;
	//删除账单中此餐台号原有的账单信息
	CString Sql1="delete from paybill where 桌号='"+zhuohao+"'";
	theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);
	double Totle=0;						//定义变量记录总消费
	//将经过增加或减少的新账单信息写入数据库
	for(int n=0;n<i;n++)
	{
		Str=m_CaidanCheck.GetItemText(n,0);			//获取第 n 行第一列的文本
		Str1=m_CaidanCheck.GetItemText(n,1);			//获取第 n 行第二列的文本
		//在菜式信息表中获取菜名一致的信息
		Sql1="select * from caishiinfo where 菜名='"+Str+"'";
		m_pRs=theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);
		//获取该菜名的菜价信息
		Value=(char*)(_bstr_t)m_pRs->GetCollect("菜价");
		//将数量与菜价转化成整型数，相乘得到总消费额
		Totle=atof(Value)*atof(Str1);
		TotleValue=(char*)(_bstr_t)Totle;			//将总消费额转化成 CString 型
		Sql1="insert into paybill(桌号,菜名,数量,消费) values('"+zhuohao+"','"+Str+"','"+Str1+"','"+TotleValue+"')";
		//将菜单信息插入数据表中
		theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);
	}
```

```
        AfxMessageBox("操作成功");
        CDialog::OnOK();
}
```

（7）单击“取消”按钮关闭当前对话框，代码如下：

```
CDialog::OnCancel();
```

2.9.4 单元测试

在加菜减菜模块中，由于在加菜过程中会先调用当前餐台号所拥有的菜式账单，用户对其添加数据确认后会重复向数据表中写入原有的菜式信息，使顾客的账单出错。

为避免上述情况的发生，在系统将新生成的菜式信息保存进数据表时，必须将数据表中的原有菜式信息全部删除，将经过加菜、减菜后的数据表作为当前餐台号的最新账单。为实现这一目的，笔者在“确定”按钮的单击事件下添加如下代码：

```
//删除所选餐台号的账单信息
CString Sql1="delete from paybill where  桌号="+zhuohao+"";
theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText);                //执行语句
```


视频讲解

2.10 结账模块设计

2.10.1 结账模块概述

结账模块可对当前顾客消费进行结算，顾客结账完成后系统自动将收入金额的数据写入数据表中，从而能很好地反映营业情况。结账模块的运行效果如图 2.38 所示。



图 2.38 结账模块的运行效果

2.10.2 结账模块技术分析

在结账时，如果顾客所在的餐台号比较靠后，在下拉列表框控件中就必须按下拉按钮逐个寻找，在结账顾客数量较多的情况下，这种方法显然严重影响了工作效率。为此笔者为下拉列表框控件增加了手动输入的功能，使营业员在结账时既可以在下拉列表框中选择桌号，也可以手动输入桌号，极大地方便了使用者，提高了结账速度和顾客的满意程度。

要实现上述功能，就必须给列表控件添加一个 EDITCHANGE 事件，在事件中添加相应代码对输入的信息进行判断。本系统中的餐台号都是 4 位数，因此在事件中首先判断输入的是不是一个 4 位数，如果不是，则提示错误信息；如果是，则显示相应的消费信息。实现这一功能需要使用 CString 类提供



的 GetLength 方法，语法如下：

```
int GetLength()
```

返回值是一个整型数，是字符串的长度。

2.10.3　结账模块实现过程

本模块使用的数据表：paybill、TableUse

（1）在 Resources 选项卡中插入一个对话框资源，为其新建一个 CJiezhangdlg 类，在类中定义一个_RecordsetPtr 类型变量 m_pRs 并导入全局变量 theApp。在“结账”对话框中添加 5 个静态文本控件、3 个编辑框控件、一个下拉列表框控件、一个列表控件和两个按钮控件。

各个控件属性及变量设置如表 2.7 所示。

表 2.7　控件属性及变量设置

控件 ID	控 件 属 性	对 应 变 量
IDC_COMBO1	Dropdown	CComboBox m_Combo
IDC_yingshou	Read-only	CEdit m_YingShou
IDC_shishou	Visible	CEdit m_ShiShou
IDC_zhaoling	Read-only	CEdit m_ZhaoLing
IDC_mingxi	Report	CListCtrl m_MingXi

（2）为对话框进行初始化设置，为类添加一个成员变量 res，类型为 Bool 型。该变量主要控制下拉列表框控件接受数据的方式，False 为下拉选择型，True 为手动输入型。

为类添加一个 WM_INITDIALOG 消息，对列表控件设置样式并对其内容进行初始化设置。代码如下：

```
BOOL CJiezhangdlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	//设置窗口图标
	SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDI_ICON_pay)),TRUE);
	CString TheValue;
	//获取数据表中正在消费的餐台号
	m_pRs=theApp.m_pCon->Execute((_bstr_t)("select * from TableUSE where TableUSEID=1"),NULL,adCmdText);
	//将餐台号添加进下拉列表框控件中
❶	if(m_pRs->GetRecordCount()==0)			//如果记录数量为 0 则返回
		return true;
	if(m_pRs->GetRecordCount()==1)			//如果记录数量为 1 则将数据添加进下拉列表框控件
	{
		//获取记录中的餐台号信息
		TheValue=(char*)(_bstr_t)m_pRs->GetCollect("桌号");
❷		m_Combo.AddString(TheValue);		//将餐台号信息添加进下拉列表框控件中
		return true;
```

```
    }
    while(!m_pRs->adoEOF)                              //当记录集不为空时
    {
        //获取餐台号信息
        TheValue=(char*)(_bstr_t)m_pRs->GetCollect("桌号");
        m_Combo.AddString(TheValue);                   //将餐台号信息添加进下拉列表框中
        m_pRs->MoveNext();                             //继续下一条记录
    }
    //设置消费明细列表样式
    m_MingXi.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_HEADERDRAGDROP|
                LVS_EX_ONECLICKACTIVATE|LVS_EX_GRIDLINES);
    m_MingXi.InsertColumn(0,"菜名",LVCFMT_LEFT,100,0); //为消费明细列表添加 3 列并分别命名
    m_MingXi.InsertColumn(1,"数量",LVCFMT_LEFT,100,1);
    m_MingXi.InsertColumn(2,"消费(元)",LVCFMT_LEFT,120,1);
    res = FALSE;                                       //下拉列表框控件获取数据的方式，默认是下拉选择型
    return true;
}
```

代码贴士

❶ GetRecordCount：该方法的返回值就是下拉列表框控件中的总行数。
❷ AddString：该方法用于向下拉列表框中插入选项。

（3）在对话框左边的控件窗口中选择下拉列表框控件，再在右边消息窗口中选择 SELCHANGE 事件。代码如下：

```
void CJiezhangdlg::OnSelchangeCombo1()
{
    UpdateData();
    CString str,sql,caiming,shuliang,xiaofei,xiaofeitotle,TheValue;
    //定义变量存放总消费金额数值
    double totle=0;
    //获得当前选择项的信息并存入变量
❶   m_Combo.GetLBText(m_Combo.GetCurSel(),str);
    sql="select * from paybill where 桌号="+str+"";
    //获取当前餐台号的账单信息
    m_pRs=theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
❷   m_MingXi.DeleteAllItems();                         //清空列表控件
    //将获取的账单信息添加进明细列表控件中
    while(m_pRs->adoEOF==0)                            //判断记录是否为空
    {
        //获取消费信息并存入变量
        TheValue=(char*)(_bstr_t)m_pRs->GetCollect("消费");
        totle+=atof(TheValue);                         //将消费转换成整型进行累加
        //获取菜名信息并存入变量
        caiming=(char*)(_bstr_t)m_pRs->GetCollect("菜名");
        //获取数量信息并存入变量
        shuliang=(char*)(_bstr_t)m_pRs->GetCollect("数量");
        //获取消费信息并存入变量
        xiaofei=(char*)(_bstr_t)m_pRs->GetCollect("消费");
```

```
        m_MingXi.InsertItem(0,"");                      //为明细列表插入一行
        m_MingXi.SetItemText(0,0,caiming);              //在该行的第 1 列添加菜名信息
        m_MingXi.SetItemText(0,1,shuliang);             //在该行的第 2 列添加数量信息
        m_MingXi.SetItemText(0,2,xiaofei);              //在该行的第 3 列添加消费信息
        m_pRs->MoveNext();                              //继续下一条记录
    }
    xiaofeitotle=(char*)(_bstr_t)totle;                 //算出消费总金额
    m_YingShou.SetWindowText(xiaofeitotle);             //将消费总金额在“应收”控件中显示
    UpdateData(false);
}
```

代码贴士

❶ GetLBText：该方法用于获取相应行的文本信息并将其存入字符变量中。

❷ DeleteAllItems：该方法用于删除列表控件中的所有数据。

（4）在对话框左边的控件窗口中选择下拉列表框控件，在右边消息窗口中选择 EDITCHANGE 事件并添加如下代码：

```
void CJiezhangdlg::OnEditchangeCombo1()
{
    m_MingXi.DeleteAllItems();                          //清空列表控件
    m_YingShou.SetWindowText("");                       //“应收”控件中数值初始化
    CString str;
    m_Combo.GetWindowText(str);                         //获取下拉列表框控件中输入的数值
    if(str.GetLength()==4)                              //判断位数，这里餐台号都是 4 位数
    {
        UpdateData();
        CString sql,caiming,shuliang,xiaofei,xiaofeitotle,TheValue;
        double totle=0;                                 //定义变量存放消费总数
        sql="select * from paybill where 桌号="+str+"";
        //在数据表中查询当前餐台号的信息
        m_pRs=theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
        while(m_pRs->adoEOF==0)                         //判断记录是否为空
        {
            //获取消费信息并存入变量
            TheValue=(char*)(_bstr_t)m_pRs->GetCollect("消费");
            totle+=atof(TheValue);                      //将消费转换成整型进行累加
            //获取菜名信息并存入变量
            caiming=(char*)(_bstr_t)m_pRs->GetCollect("菜名");
            //获取数量信息并存入变量
            shuliang=(char*)(_bstr_t)m_pRs->GetCollect("数量");
            //获取消费信息并存入变量
            xiaofei=(char*)(_bstr_t)m_pRs->GetCollect("消费");
            m_MingXi.InsertItem(0,"");                  //为明细列表插入一行
            m_MingXi.SetItemText(0,0,caiming);          //在该行的第 1 列添加菜名信息
            m_MingXi.SetItemText(0,1,shuliang);         //在该行的第 2 列添加数量信息
            m_MingXi.SetItemText(0,2,xiaofei);          //在该行的第 3 列添加消费信息
            m_pRs->MoveNext();                          //继续下一条记录
        }
```

```
            xiaofeitotle=(char*)(_bstr_t)totle;               //算出消费总金额
            //将消费总金额显示在“应收”控件中
            m_YingShou.SetWindowText(xiaofeitotle);
            res = TRUE;                                       //表示是用手动输入方式
            UpdateData(false);
        }
}
```

（5）在顾客付款后，应在“实收”编辑框中输入顾客的付款金额，此时“找零”编辑框中应该实时计算出应找给顾客的金额。

为达到上述目的，先在对话框中选中对应“实收”的编辑框控件名，再在右边选择它的 EN_CHANGE 事件，在此事件中添加如下代码：

```
void CJiezhangdlg::OnChangeEDITshishou()
{
        double zhaoling;
        CString ShiShou,YingShou;
        m_ShiShou.GetWindowText(ShiShou);                     //获得实收的金额数
        m_YingShou.GetWindowText(YingShou);                   //获得应收的金额数
        zhaoling = atof(ShiShou) - atof(YingShou);            //算出应该找给顾客的金额数
        CString str;
        str.Format("%0.2f",zhaoling);                         //将找零格式化为两位小数
        m_ZhaoLing.SetWindowText(str);                        //将找零实时显示在编辑框中
}
```

（6）在单击“结账”按钮时，系统自动将当前餐台号的使用状态变成空闲状态，并将账单数据表清空，然后将这次结账的收入写进日收入数据表中，方便查询日收入，代码如下：

```
UpdateData();
CString str,str1,str2,str3;
CString TheValue;
CString ShiShou,YingShou;
m_Combo.GetWindowText(str1);                                  //获取下拉列表框中的餐台号信息
if(str1.GetLength()<4||str1.GetLength()>4)                    //判断餐台号是否为 4 位数
{
        AfxMessageBox("输入错误");                            //如果不是 4 位数则提示出错
        return;
}
CString bjsql="select * from TableUSE where  桌号='"+str1+"'";
//查询数据表中对应的餐台号信息
m_pRs=theApp.m_pCon->Execute((_bstr_t)bjsql,NULL,adCmdText);
if(m_pRs->adoEOF)                                             //判断记录是否为空
{
        AfxMessageBox("没有这张餐台");                        //如果为空，则提示没有餐台
        return;
}
//获取对应餐台号的使用情况
CString bjstr=(char*)(_bstr_t)m_pRs->GetCollect("TableUSEID");
```

```
if(bjstr=="0")                                          //如果为 0，则提示不需要付款
{
      AfxMessageBox("该桌不需要付款");
      return;
}
m_ShiShou.GetWindowText(str3);                          //获取客人付款的金额
if(str3.IsEmpty())                                      //如果编辑框为空，则提示输入
{
      AfxMessageBox("请输入顾客付款");
      return;
}
if(res == TRUE)                     //判断 res 的状态，TRUE 则为手动输入型，FALSE 为下拉选择型
      m_Combo.GetWindowText(str);                       //手动输入获取编辑框中的文本信息
else
      m_Combo.GetLBText(m_Combo.GetCurSel(),str);       //获取下拉列表框中的内容
m_ZhaoLing.SetWindowText("");                           //“找零”编辑框初始化显示
double zhaoling,rishouru=0;
m_ShiShou.GetWindowText(ShiShou);                       //获取实收金额
m_YingShou.GetWindowText(YingShou);                     //获取应收金额
rishouru=atof(YingShou);                                //将应收的金额赋值给日收入
if(atof(ShiShou)<atof(YingShou))                        //判断实收金额和应收金额的大小
{
      AfxMessageBox("想吃霸王餐？");                     //如果实收小于应收则提示
      return;
}
else
{
      CTime time;                                       //定义一个时间类变量
      time = CTime::GetCurrentTime();                   //获取当前系统时间
      CString str1 = time.Format("%Y-%m-%d");           //将系统时间转换成 CString 型变量
      zhaoling=atof(ShiShou)-atof(YingShou);            //算出找零金额
      TheValue=(char*)(_bstr_t)zhaoling;                //将找零金额转换成 CString 型变量
      m_ZhaoLing.SetWindowText(TheValue);               //在“找零”编辑框中显示找零金额
      UpdateData(false);
      CString sql;
      str2="update TableUSE set TableUSEID=0 where 桌号="+str+" ";
      //修改付款后该桌的使用状态
      theApp.m_pCon->Execute((_bstr_t)str2,NULL,adCmdText);
      TheValue.Format("%0.2f",rishouru);                //将日收入转换成两位单精度数
      sql="update shouru set 日收入=日收入+'"+TheValue+"' where 时间='"+str1+"'";
      //将当天的日收入进行累加
      theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
      m_YingShou.SetWindowText("");                     //“应收”编辑框初始化显示
      m_ShiShou.SetWindowText("");                      //“实收”编辑框初始化显示
      m_ZhaoLing.SetWindowText("");                     //“找零”编辑框初始化显示
      m_Combo.SetWindowText("");                        //下拉列表框初始化显示
      m_Combo.DeleteString(m_Combo.GetCurSel());        //删除下拉列表框中所选中的选项
      m_MingXi.DeleteAllItems();                        //清空列表控件
      sql="delete from paybill where 桌号="+str+"";
```

```
        //将账单中该桌的信息删除
        theApp.m_pCon->Execute((_bstr_t)sql,NULL,adCmdText);
        AfxMessageBox("欢迎再来");
}
```

（7）给“再见”按钮添加如下代码：

```
CDialog::OnCancel();
```

2.10.4　单元测试

在结账模块中，如果在下拉列表框控件中没有任何数据被输入或选择时，直接单击了“结账”按钮或误按了 Enter 键，将会出现如图 2.39 所示的提示信息。

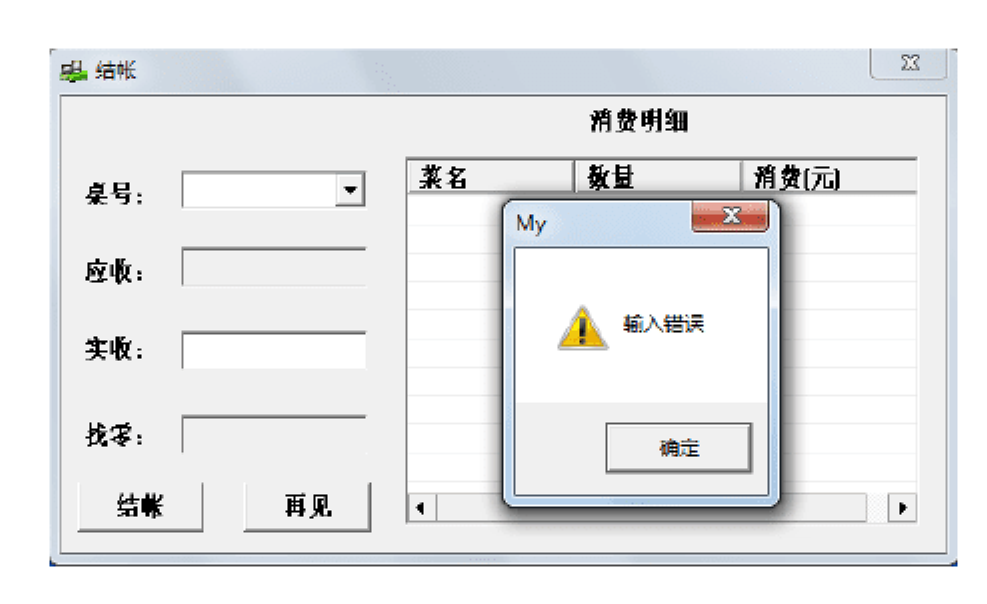


图 2.39　结账模块提示出错

导致这种错误的原因主要是系统没有对下拉列表框控件的输入进行判断，如果下拉列表框控件的属性是 Drop List，则只需判断它的当前项的值是否等于−1 即可，−1 表示没有选择，代码如下：

```
if(m_Combo.GetCurSel()==-1)
{
        AfxMessageBox("输入错误");
        return;
}
```

但在本系统中，由于列表控件既要接受下拉选择，又要接受文本输入，所以将其属性设置为 DropDown。在判断其是否有数值输入时需要判断输入的位数，因为只有当输入位数与餐台号位数相等时，系统才通过验证，代码如下：

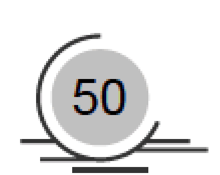
```
if(str1.GetLength()<4||str1.GetLength()>4)
{
        AfxMessageBox("输入错误");
        return;
}
```



2.11　数据库维护模块设计

2.11.1　数据库维护模块概述

在系统的日常使用过程中，数据库损坏或数据库丢失的现象时有发生，为了避免该现象对用户造





成影响，本系统中加入了数据库维护模块，用户可以通过该模块对数据库进行备份、还原及初始化等操作，大大提高了用户数据的安全性。数据库维护模块的运行效果如图 2.40 和图 2.41 所示。

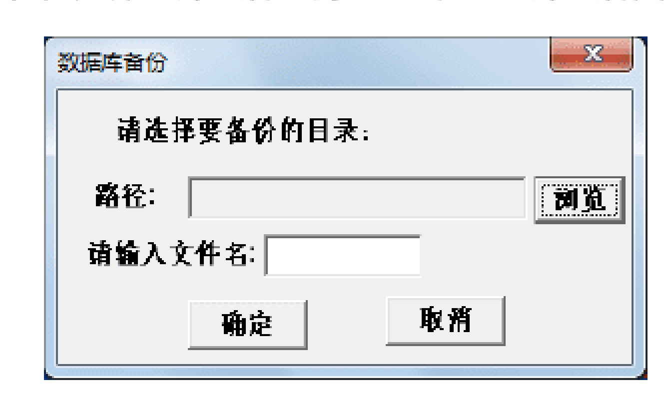


图 2.40　数据库备份运行效果

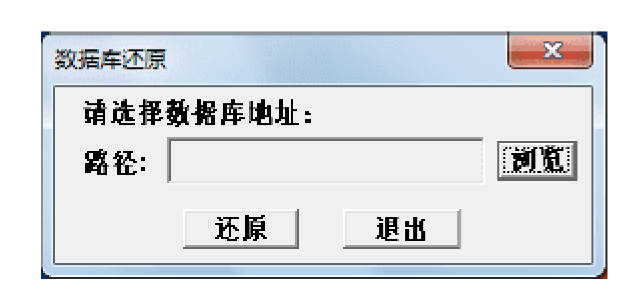


图 2.41　数据库还原运行效果

2.11.2　数据库维护模块技术分析

由于本系统采用的是 Access 2010 数据库，所以在数据库的操作方面与 SQL Server 数据库有一定的不同，例如，数据库的备份操作，SQL Server 数据库直接使用 Backup 语句即可实现，而 Access 数据库中没有 Backup 语句供程序员使用。在 Access 数据库中备份、还原数据库的方法其实与读者在 Windows 中备份文件的方法一样。在 Access 数据库中，备份数据库就是将源数据库复制到相应文件夹的过程，而还原数据库则是备份操作的逆操作，即将以前备份好的数据库复制并粘贴到现在数据库所在的文件夹中，对现有数据库进行覆盖操作以达到还原的目的。以备份操作为例，在系统进行复制前首先要获得当前数据库所在的位置，这时可以用 GetCurrentDirectory 方法获取当前数据库所在的地址路径。

```
char buf[256];                          //首先创建一个字符数组存放路径
::GetCurrentDirectory(256,buf);         //获取数据库所在的文件夹
strcat(buf,"\\canyin.mdb");             //将文件夹路径与数据库名称连接组成数据库的地址路径
```

2.11.3　数据库维护模块实现过程

1．数据库备份

数据库备份的具体实现过程如下：

（1）在 Resources 选项卡中插入一个对话框资源，为其新建一个 CCopydlg 类，在对话框中添加 3 个静态文本控件、两个编辑框控件和 3 个按钮控件。将“路径”编辑框控件的属性设置为 Read-only，对其添加一个 CEdit 类型变量 m_Edit；再对“请输入文件名”编辑框添加一个 CString 类型变量 m_Name。

（2）在单击“浏览”按钮时，弹出一个文件路径选择对话框方便用户选择备份路径，代码如下：

```
CString ReturnPach;                     //定义一个字符串变量保存存储路径
TCHAR szPath[_MAX_PATH];
BROWSEINFO bi;                          //定义一个对话框实例
bi.hwndOwner=NULL;
bi.pidlRoot=NULL;
bi.lpszTitle=_T("请选择备份文件夹");     //设置窗口标题
bi.pszDisplayName=szPath;
bi.ulFlags=BIF_RETURNONLYFSDIRS;
bi.lpfn=NULL;
```

```
bi.lParam=NULL;
LPITEMIDLIST pItemIDList=SHBrowseForFolder(&bi);        //获得选择路径
if(pItemIDList)                                          //判断路径是否为空
{
	if(SHGetPathFromIDList(pItemIDList,szPath))
		ReturnPach=szPath;                               //如果路径不为空，则将路径赋给字符串
}
else
	ReturnPach="";                                       //路径为空，字符串同时为空
m_Edit.SetWindowText(ReturnPach);                        //将路径显示在编辑框中
```

（3）在添加完想要保存的路径后，用户要在编辑框中输入想保存的文件名称，随后给“确定”按钮添加如下代码：

```
UpdateData();
CString str,strpath;
m_Edit.GetWindowText(str);                               //获取编辑框中的路径地址
strpath = str+"\\"+m_Name+".mdb";                        //将路径和要求的文件名以固定的格式组合
char buf[256];
::GetCurrentDirectory(256,buf);
strcat(buf,"\\canyin.mdb");                              //获取当前程序的数据库地址
CopyFile(buf,strpath,false);                             //复制文件，false 代表遇到同文件名进行覆盖
MessageBox("备份完成！ ","系统提示",MB_OK|MB_ICONEXCLAMATION);
CDialog::OnOK();
```

2．数据库还原

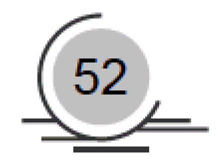
Access 数据库的还原操作其实就是备份操作的一个逆过程，备份操作是将原有数据库复制到指定文件夹，而还原操作则是将指定文件夹中的数据库文件复制到当前数据库文件夹中并进行覆盖，从而实现数据库的还原。

（1）在 Resources 选项卡中插入一个对话框资源，为其新建一个 CReturndlg 类，在对话框中添加两个静态文本控件、一个编辑框控件和 3 个按钮控件。将“路径”编辑框的属性设置为 Read-only 并为其添加一个 CEdit 类型变量 m_Edit。

（2）在“还原”对话框中单击“浏览”按钮后，需要显示文件目录中某个文件，代码如下：

```
CFileDialog dlg(TRUE,"mdb",NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
		(*.mdb)|*.mdb",NULL);           //创建一个文件对话框并且只显示文件后缀名为.mdb 的文件
if(dlg.DoModal()==IDOK)                                  //弹出文件对话框判断是否单击 OK 按钮
{
	CString str;
	str = dlg.GetPathName();                             //获取选择的文件路径
	m_Edit.SetWindowText(str);                           //将路径添加至编辑框控件中
}
```

（3）在进行数据库还原操作前，系统自动判断当前程序数据库存放地址路径，以便用户复制数据库文件。为类添加一个 WM_INITDIAOLG 消息，代码如下：



```
BOOL CReturndlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	::GetCurrentDirectory(256,buf);
	strcat(buf,"\\canyin.mdb");                           //获取当前数据库路径地址
	return TRUE;
}
```

（4）单击“还原”按钮时，系统自动将用户选取的数据库文件复制到当前数据库所在文件，代码如下：

```
UpdateData();
CString str;
m_Edit.GetWindowText(str);                                //获取需还原的数据库路径
CopyFile(str,buf,false);                                  //将数据库文件复制到源数据库文件
MessageBox("还原完成！","系统提示",MB_OK|MB_ICONEXCLAMATION);
CDialog::OnOK();
```

3．数据库初始化

当数据库中存储的信息已经失效时，手动删除数据无疑增加了用户的工作量。为减轻用户的工作量，本系统添加了数据库初始化功能，执行该功能后将清空除用户信息表外其他所有数据表中的数据。为菜单中的“数据库初始化”菜单项添加响应事件代码：

```
//弹出窗口，确认是否要执行命令
if(MessageBox("确定要初始化数据库吗?","提示",MB_YESNO)==IDYES)
{
	CString Sql1="delete from caishiinfo";                //删除菜式信息表中的内容
	CString Sql2="delete from jinhuo";                    //删除进货信息表中的内容
	CString Sql3="delete from shangpininfo";              //删除商品信息表中的内容
	CString Sql4="delete from shouru";                    //删除收入信息表中的内容
	CString Sql5="delete from paybill";                   //删除账单信息表中的内容
	theApp.m_pCon->Execute((_bstr_t)Sql1,NULL,adCmdText); //执行第 1 条数据库语句
	theApp.m_pCon->Execute((_bstr_t)Sql2,NULL,adCmdText); //执行第 2 条数据库语句
	theApp.m_pCon->Execute((_bstr_t)Sql3,NULL,adCmdText); //执行第 3 条数据库语句
	theApp.m_pCon->Execute((_bstr_t)Sql4,NULL,adCmdText); //执行第 4 条数据库语句
	theApp.m_pCon->Execute((_bstr_t)Sql5,NULL,adCmdText); //执行第 5 条数据库语句
	AfxMessageBox("初始化成功");
	return;
}
else
	return;
```

2.11.4　单元测试

在数据库还原过程中，有时会发生数据库不能正常还原到源数据库目录的情况，主要原因是将 GetCurrentDirectory 语句放在了“还原”按钮中，而 GetCurrentDirectory 语句的作用是获取当前目录，

如果在数据库还原操作前进行了数据库备份操作，那么再次执行还原操作的结果就将文件直接复制到了刚刚备份操作所选择的目录中。

为解决这一问题，笔者在数据库还原类中定义了一个 OnInitDialog 消息，在打开还原窗口时，系统将通过这个消息初始化对话框找到正确的数据库目录，代码如下：

```
BOOL CReturndlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	::GetCurrentDirectory(256,buf);                    //获取正确的数据库文件路径地址
	strcat(buf,"\\canyin.mdb");
	return TRUE;
}
```

2.12 打 包 发 行

在完成应用软件的开发工作以后，还要对项目进行打包发行。作为一款好的产品，既要保证高质量又要有一个好的包装，只有这样才能使自己的软件产品在众多产品中脱颖而出，提高竞争力。

2.12.1 选择合适的打包工具

打包是采用一系列的方法和手段把应用程序与相关的文件集中起来，形成一个可执行的程序包的过程。制作出来的程序包需要满足可执行性、简单性和可靠性等基本要求。

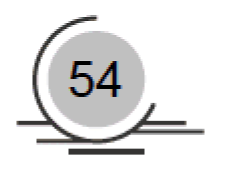
- ☑ 可执行性：程序包必须满足的核心要求，具体指制作的程序包在经过安装后可以在目的计算机上运行。
- ☑ 简单性：指操作的简化，对于安装程序来说，就是无需复杂的操作就能将应用程序安装到目的计算机上，并且可以使应用程序正常地运行。
- ☑ 可靠性：在应用程序的安装过程中，可能要对系统做某些修改，也有可能对不同的系统进行不同的调整，这就需要安装程序能识别各种环境，并采用不同的安装包进行安装，同时也必须检查环境是否满足要求。

任何一款软件都离不开安装程序，所以选择适合的打包工具进行打包就变得尤为重要。InstallShield 就是一款非常好的打包工具，它以功能强大、灵活性好、容易扩展和强大的网络支持著称，而且内建的脚本语言 InstallScript 使用户可以像使用其他语言那样制作出自己的安装脚本程序，因此成为当今流行的打包工具。

2.12.2 InstallShield 打包方案

使用 InstallShield 创建工程的步骤如下：

（1）启动 InstallShield 程序，在操作系统的任务栏中单击“开始”按钮，选择“程序”→InstallShield Microsoft Visual C++ 6.0 命令，弹出 InstallShield 窗口，如图 2.42 所示。



（2）双击 InstallShield 程序界面中的 Project Wizard 图标，弹出 Welcome 对话框，如图 2.43 所示。

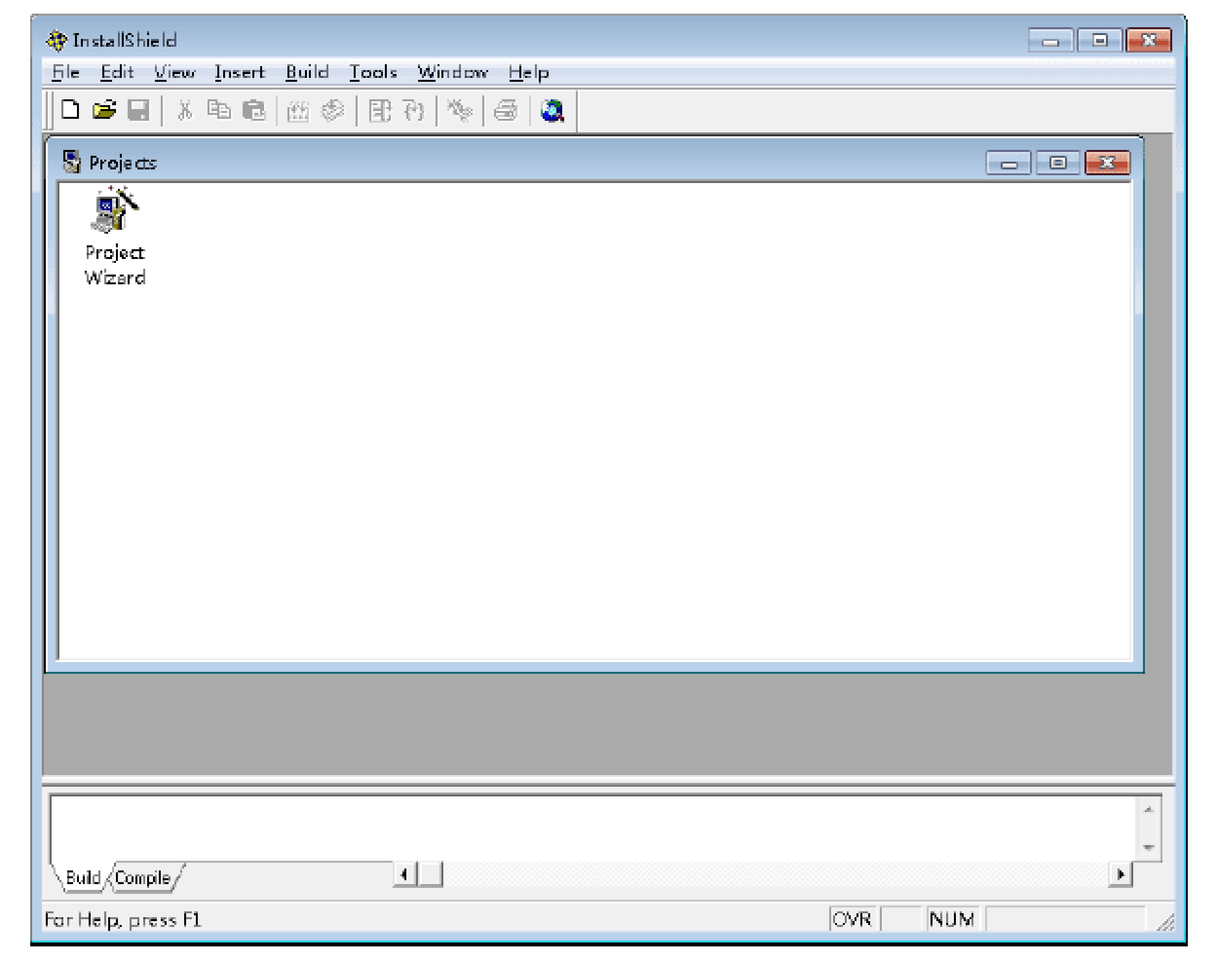


图 2.42　InstallShield 窗口



图 2.43　Welcome 对话框

（3）在 Welcome 对话框中要求用户输入如下信息。

☑　Application：应用程序名。

☑　Company Name：公司名称。

☑　Development：程序开发环境。

☑　Application：应用程序类型。

☑　Application Version：版本号。

☑　Application Executable：应用程序可执行文件。

（4）添加信息后，单击“下一步”按钮，弹出 Choose Dialogs 对话框，如图 2.44 所示。

（5）按照软件的默认设置，继续单击“下一步”按钮，直到弹出 Summary 对话框，如图 2.45 所示。

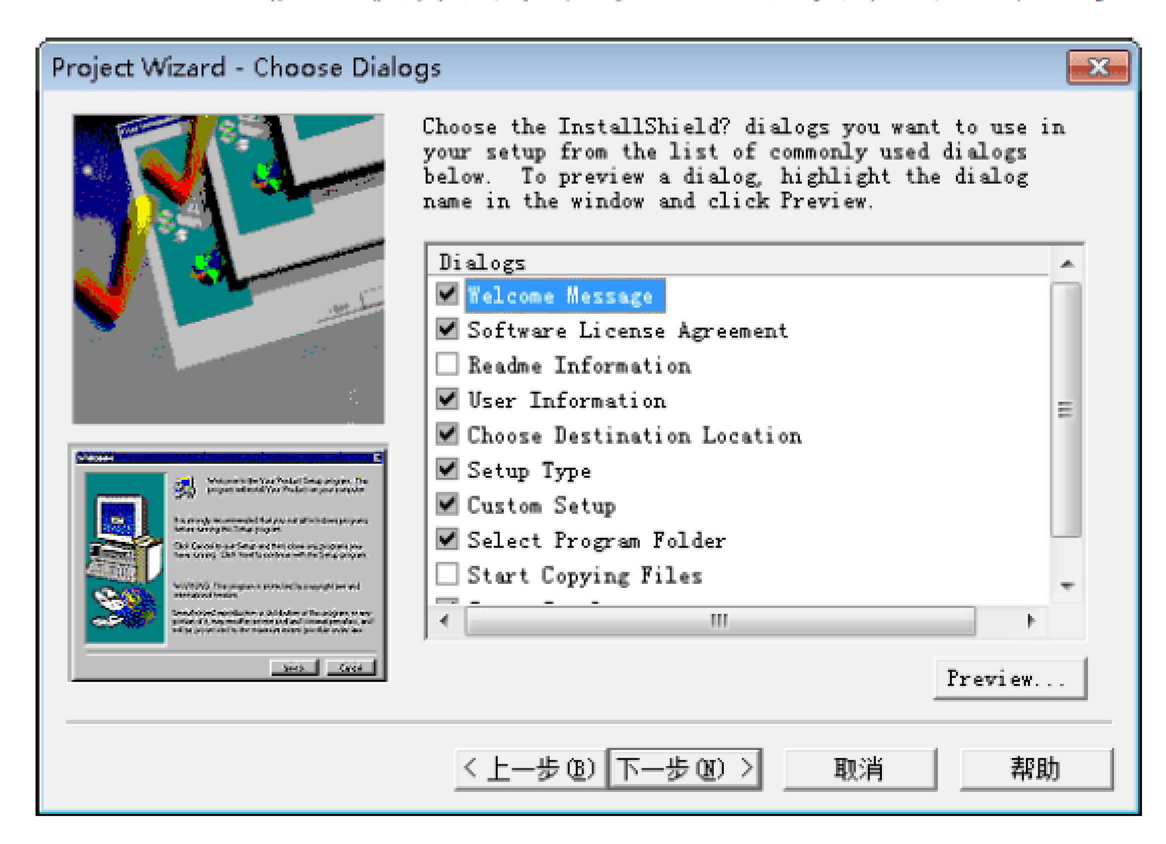


图 2.44　Choose Dialogs 对话框

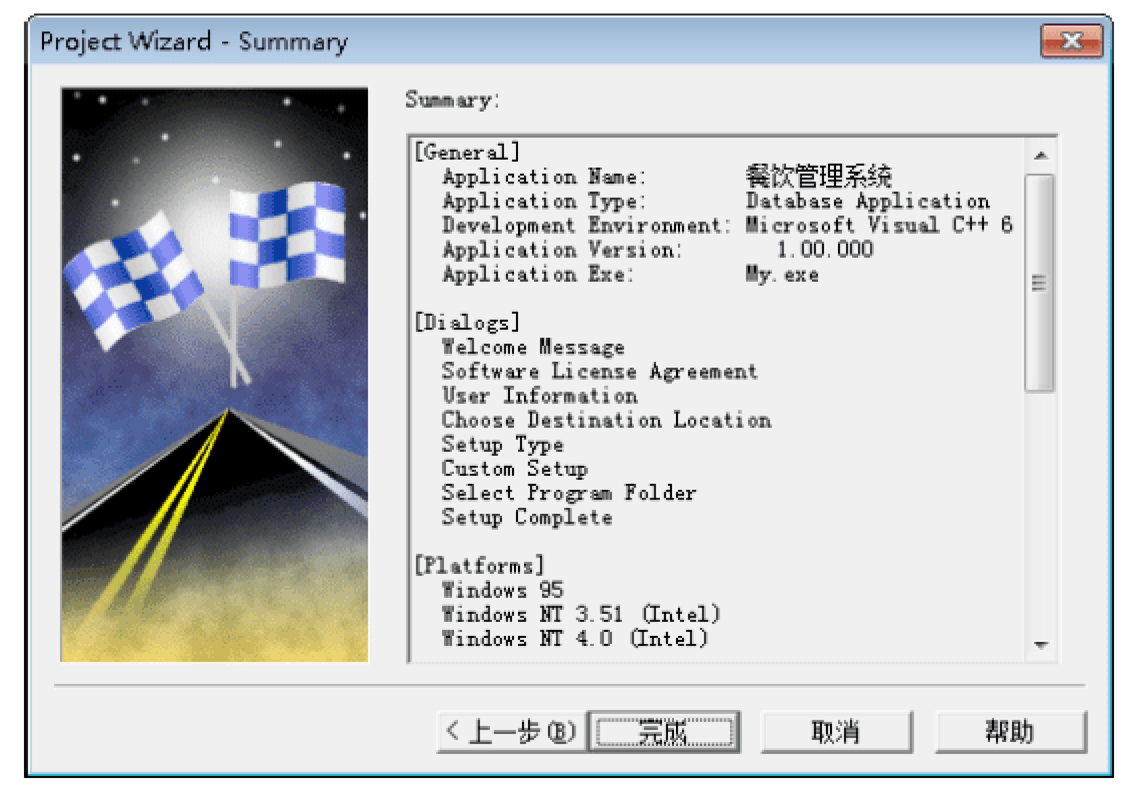
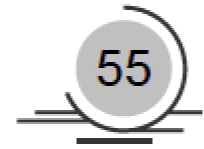


图 2.45　Summary 对话框

在 Summary 对话框中列出了用户设置的所有信息，单击“完成”按钮，创建工程，并根据用户设



置的信息生成相应的代码。

接下来才正式进入了 InstallShield 工程界面，如图 2.46 所示。

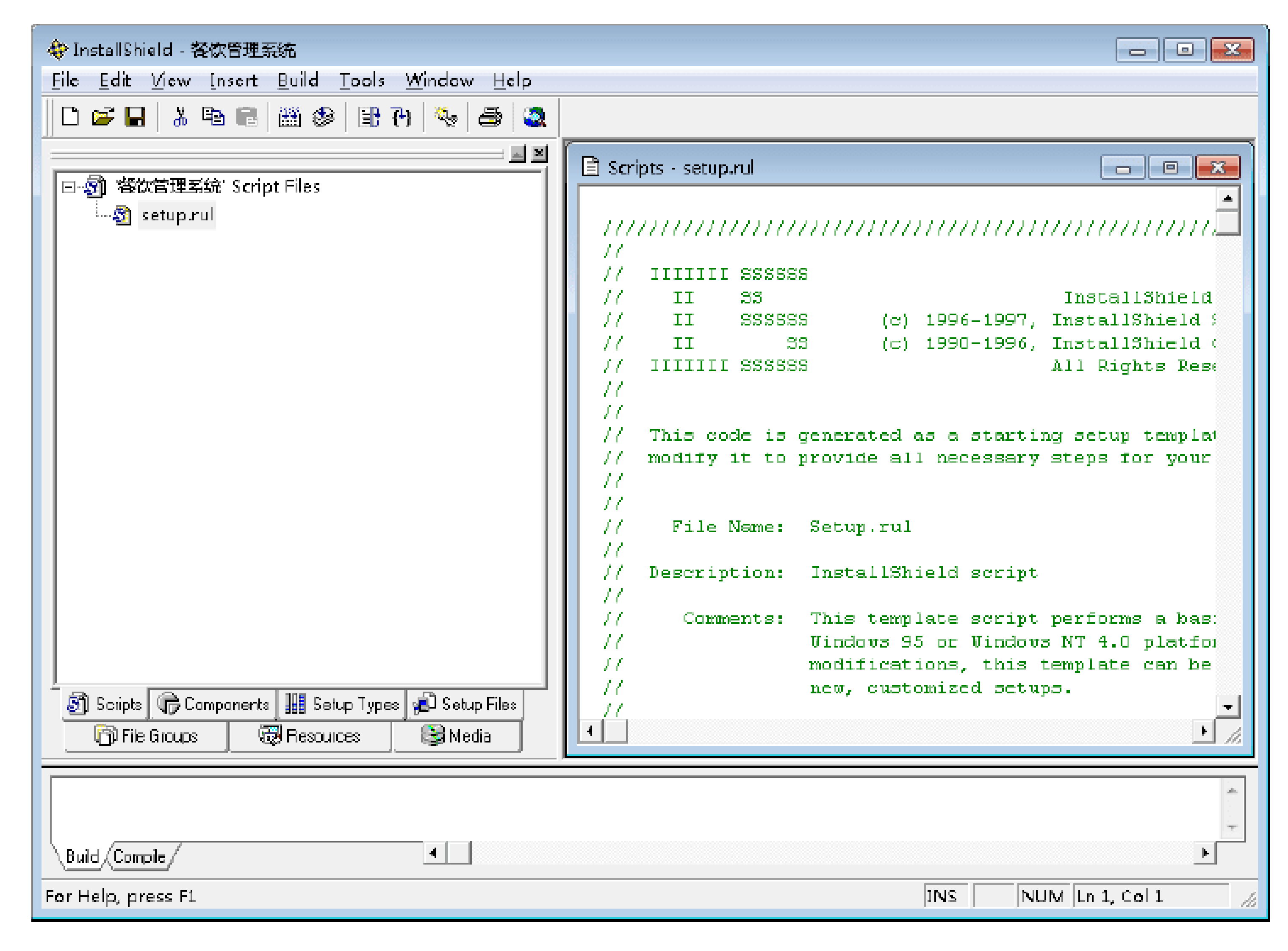


图 2.46　InstallShield 工程界面

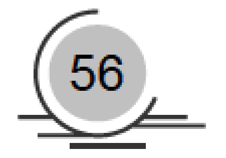
InstallShield 工程界面由标题栏、菜单栏、工具栏、工作区窗口、文档窗口、输出窗口和状态栏组成。

虽然脚本代码中生成了相关的函数，但是有些函数只有相应的框架而没有函数的具体实现。例如，脚本代码中的 DialogShowSdRegisterUserEx 函数，该函数用来对用户在安装过程中输入的序列号进行验证，但是该函数本身并没有实现这一功能，该功能需要用户自己编写，代码如下：

```
function DialogShowSdRegisterUserEx()
    NUMBER nResult;
    STRING szTitle, szMsg;
 begin

  svName = "";
  svCompany = "";

  szTitle = "";
  szMsg = "";
  dRegister:
  nResult = SdRegisterUserEx(szTitle, szMsg, svName, svCompany, svSerial);
    if(nResult == NEXT) then
        if(svSerial != "0000-1111-2222-3333") then //判断输入的序列号是否正确
            MessageBox("输入的序列号不正确！",WARNING);
            goto dRegister;
        endif;
    endif;
 return nResult;
 end;
```





完成以上代码，默认的序列号为 0000-1111-2222-3333，只有在安装程序时输入正确的序列号的用户才能顺利地进行安装。

2.12.3　设置工程文件

为了使向导生成的框架工程能够安装应用程序，还需要通过 InstallShield 开发环境提供的各种操作向导对工程进行设置，以使安装程序能够完成数据文件的复制、安装及添加快捷方式等功能。步骤如下：

（1）选择 File Groups 选项卡，展开要添加文件的文件组，在该文件组的 Links 节点处右击，在弹出的快捷菜单中选择 Insert Files 命令，将要添加的文件加入到相应的文件组，如图 2.47 所示。

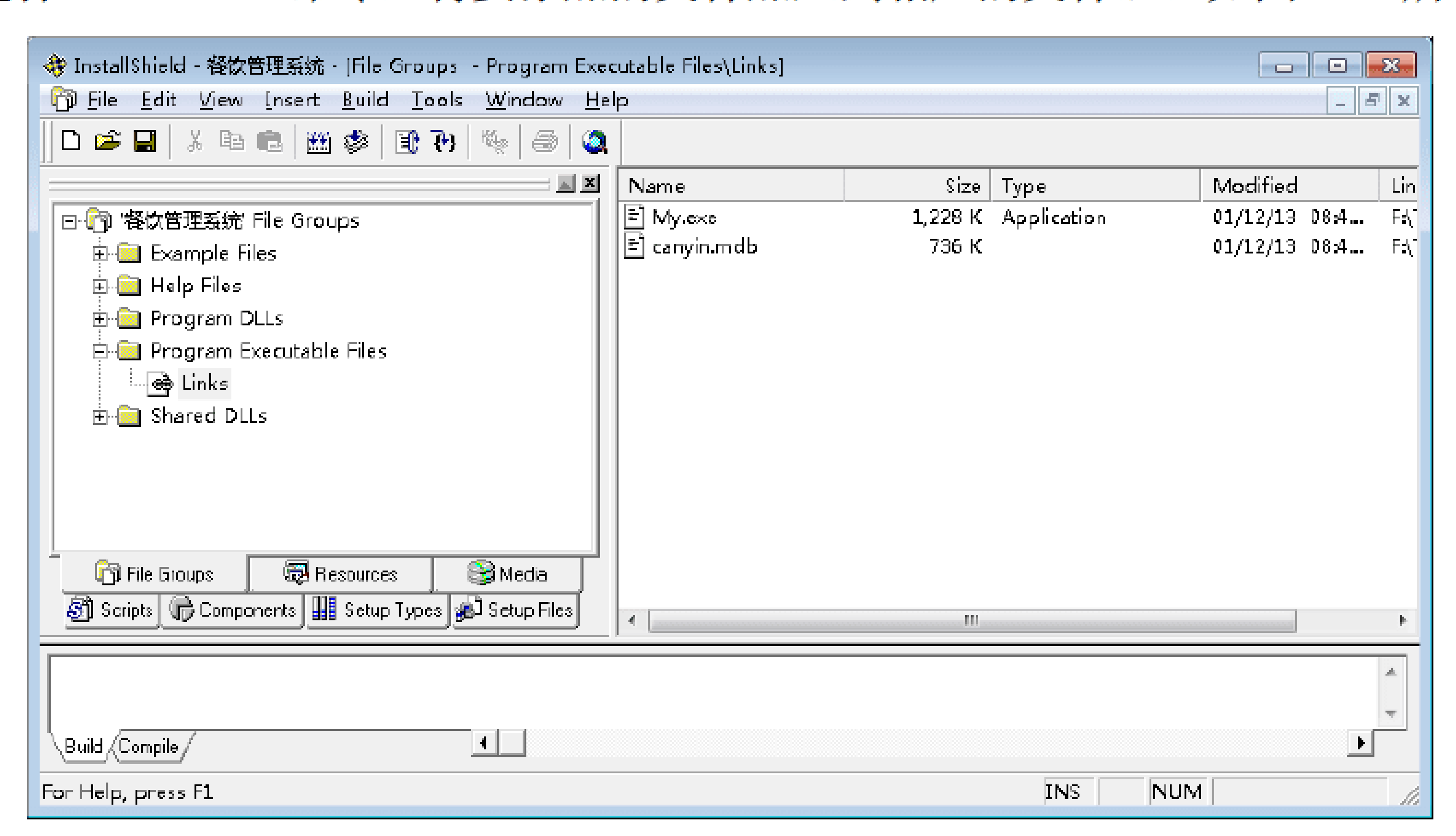


图 2.47　添加需要的文件

（2）选择 Components 选项卡，选中一个组件项，在右侧的视图中会显示该组件项的相关属性，双击 Included File Groups 选项，弹出 Properties 对话框，如图 2.48 所示。

（3）单击 Add 按钮，弹出 Add File Group 对话框，如图 2.49 所示。

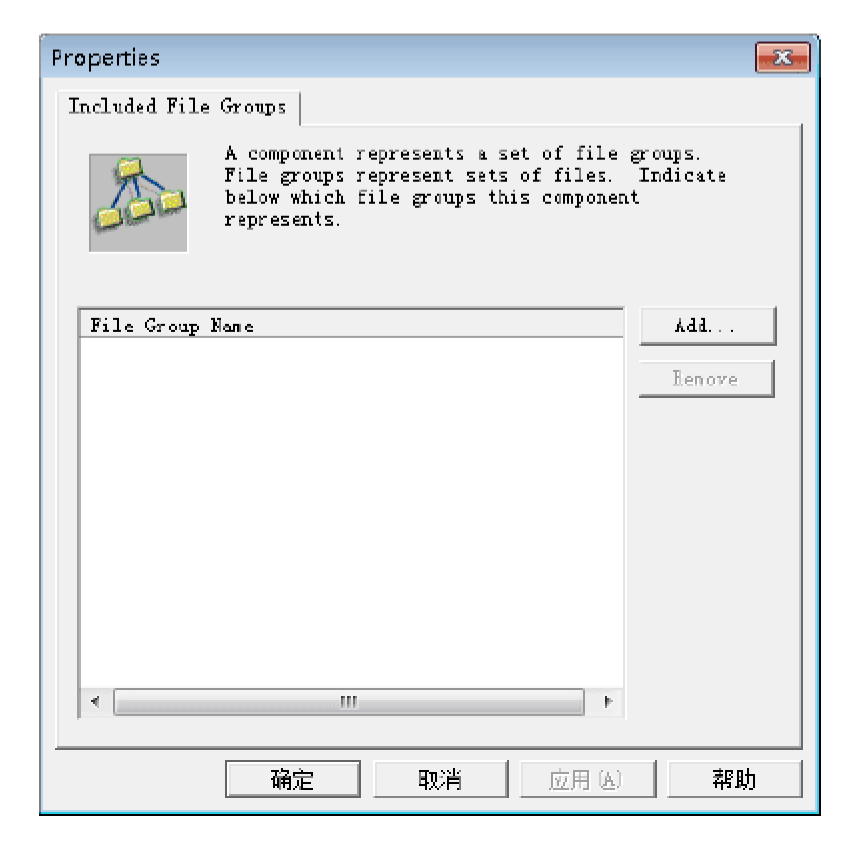


图 2.48　Properties 对话框

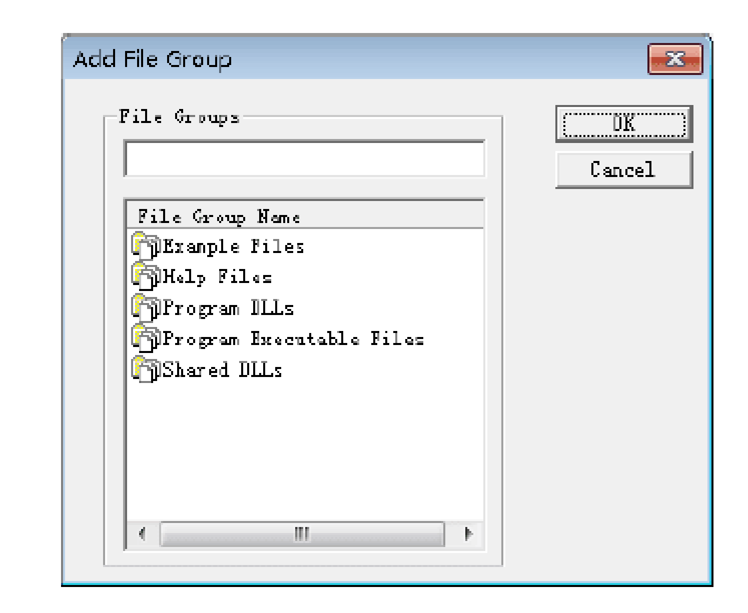


图 2.49　Add File Group 对话框

（4）选择与该组件项相关的文件组，单击 OK 按钮进行添加。

2.12.4 程序发布

（1）通过 Media 选项卡中的 Media Build Wizard 项来完成程序的发布。双击 Media Build Wizard 选项，弹出 Media Name 对话框，在 Media Name 文本框中为程序命名，如图 2.50 所示。

（2）单击“下一步”按钮，打开 Disk Type 对话框，列表框中列出了可以使用的所有发布介质，用户可以根据需要选择软盘或光盘等不同的介质，如图 2.51 所示。

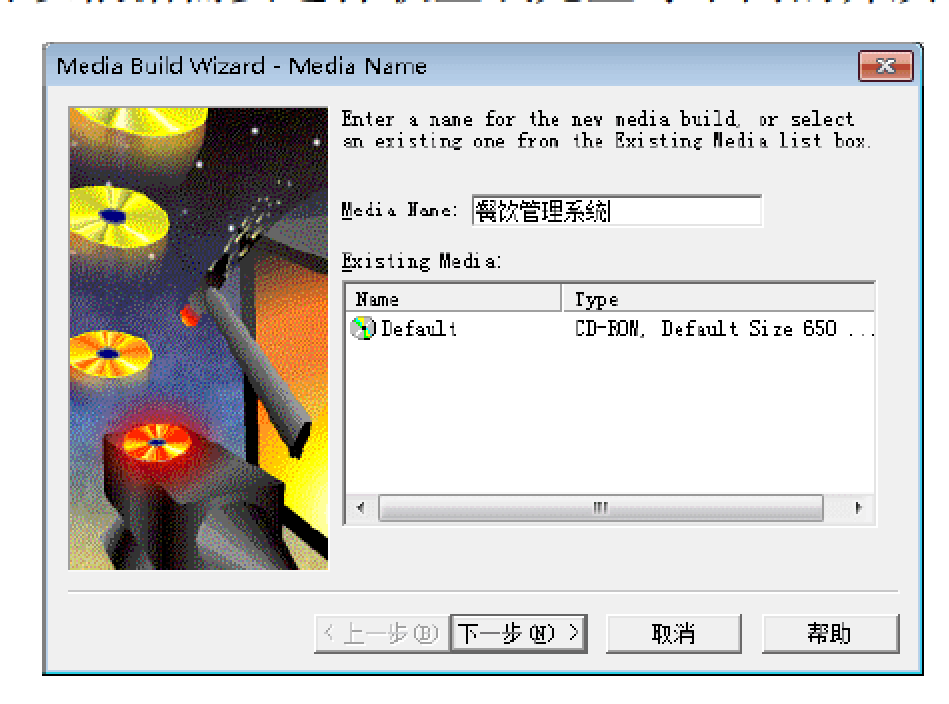


图 2.50 Media Name 对话框

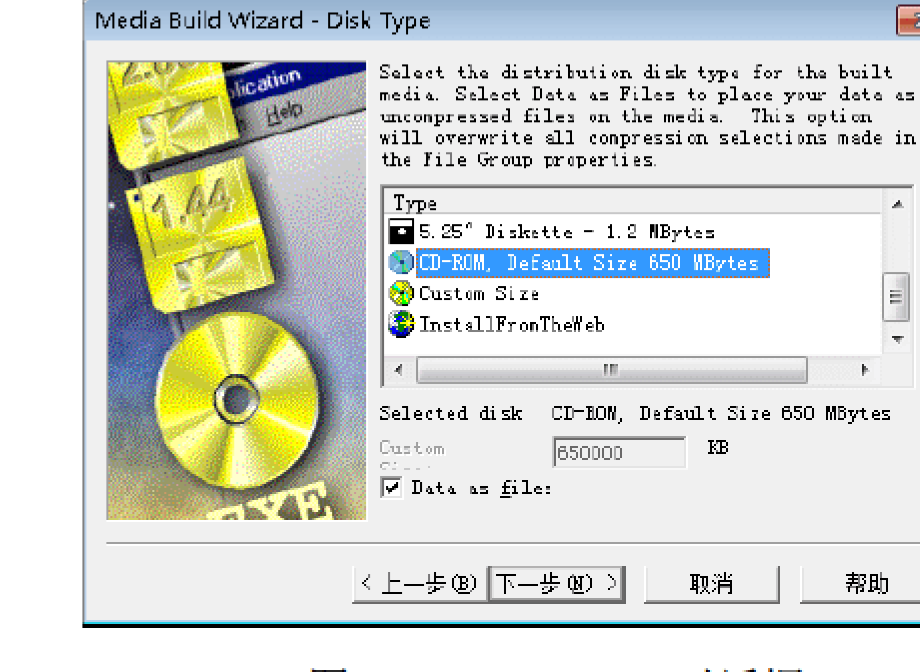


图 2.51 Disk Type 对话框

（3）单击“下一步”按钮，进入 Build Type 对话框，如图 2.52 所示。

☑ Full Build 单选按钮：创建全部所需要的文件。

☑ Quick Build 单选按钮：测试程序能否按预期的方式运行。

☑ Advanced 按钮：可以设置文件的时间、安装的路径和密码等属性。

（4）单击“下一步”按钮，进入 Tag File 对话框，如图 2.53 所示。

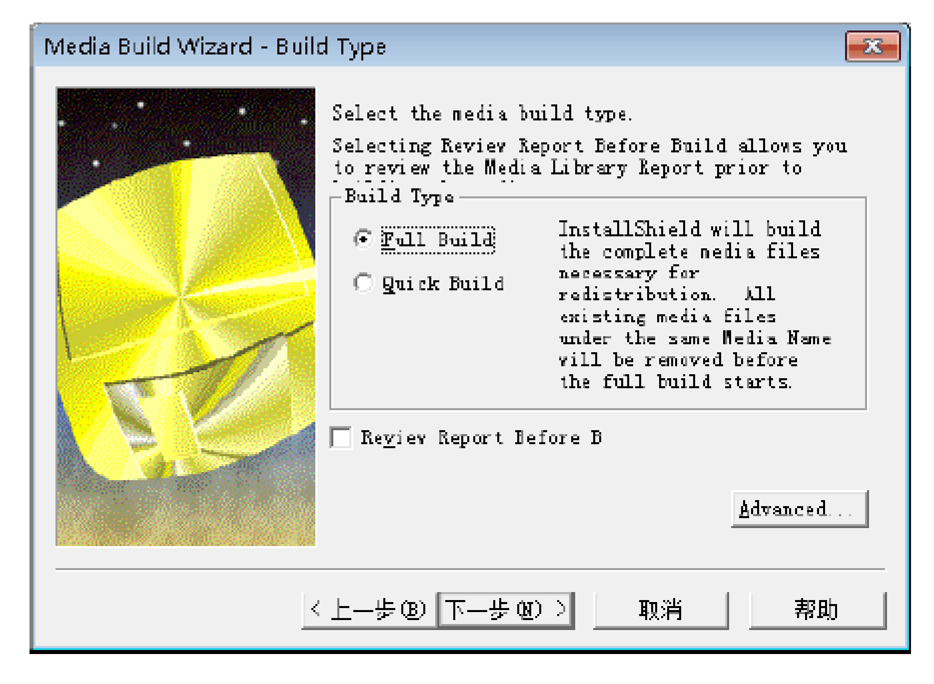


图 2.52 Build Type 对话框

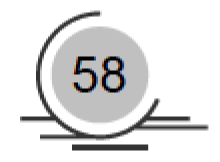


图 2.53 Tag File 对话框

（5）在 Tag File 对话框中可以设置公司名称、应用程序名以及版本等信息，单击“下一步”按钮，进入 Platforms 对话框，如图 2.54 所示。

（6）在 Platforms 对话框中可以选择使用的平台，单击“下一步”按钮，进入 Summary 对话框，如图 2.55 所示。

（7）在 Summary 对话框中列出了设置发布的所有信息，单击“完成”按钮，根据上述设置创建发布媒介，并弹出 Building Media 对话框，如图 2.56 所示。





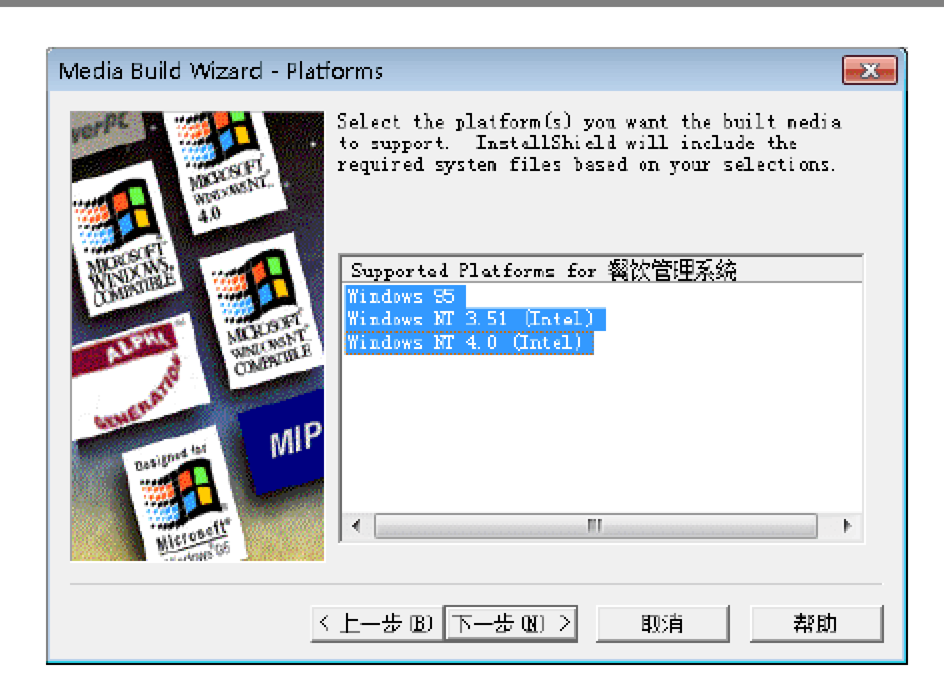


图 2.54　Platforms 对话框

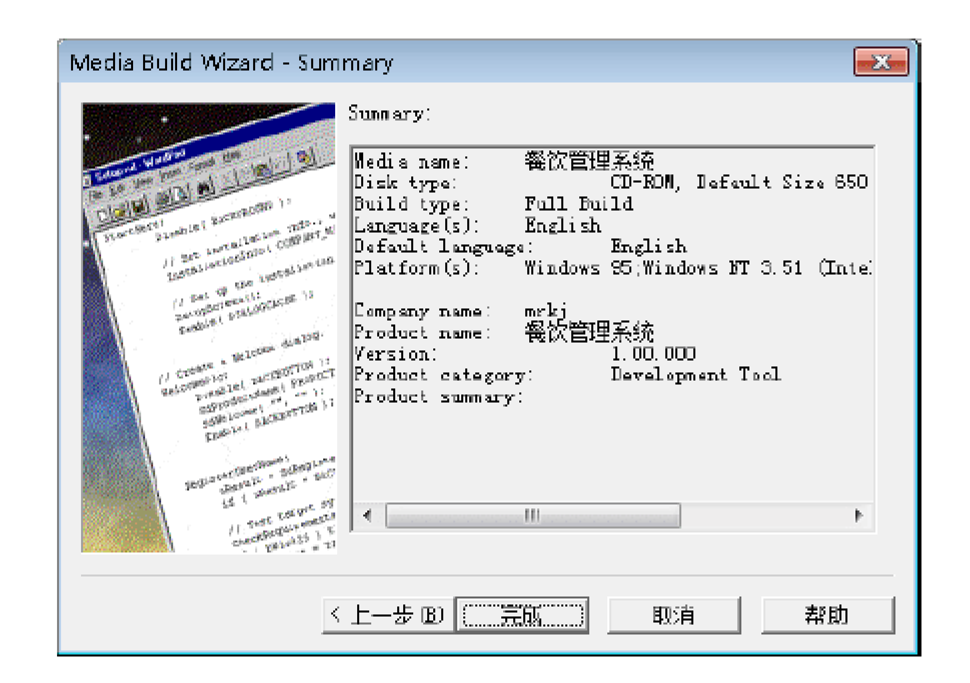


图 2.55　Summary 对话框

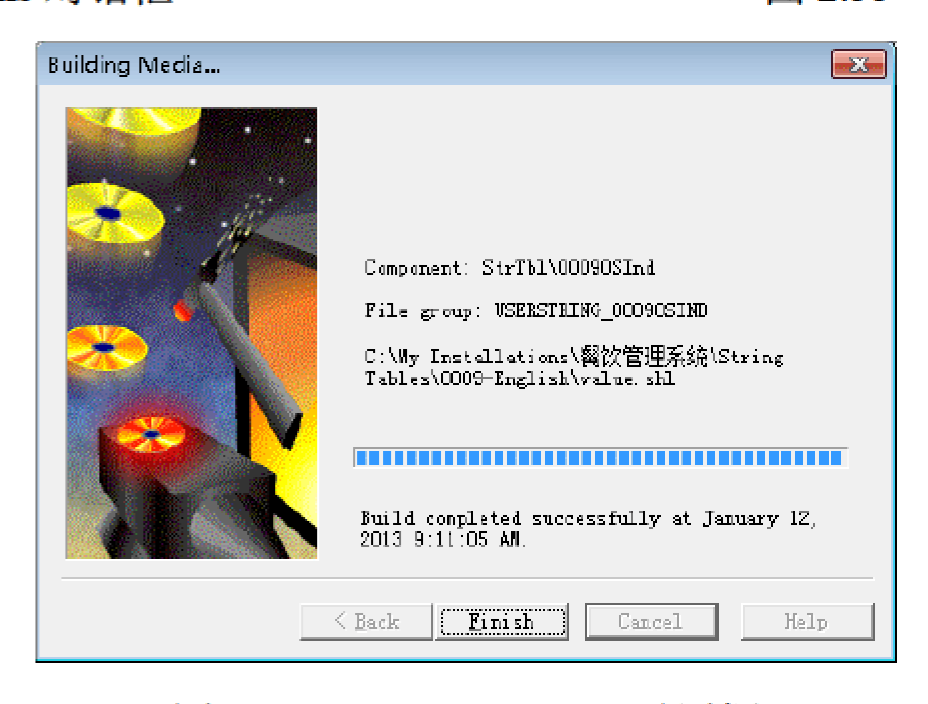


图 2.56　Building Media 对话框

（8）单击 Finish 按钮，完成程序的发布。

2.13　开发问题解析

在本章系统的开发过程中，笔者为工具栏按钮添加了鼠标提示功能，如图 2.57 所示。这样能使用户更方便地获取工具栏信息。



图 2.57　带提示功能的工具栏

要实现这一功能，需要处理 TTN_NEEDTEXT 消息的响应函数 OnToolTipNotify，通过该函数的参数可以获得工具栏按钮的 ID，从而根据 ID 获得提示信息文本。

（1）在对话框的 OnInitDialog 方法中创建工具栏窗口和图像列表窗口，关联图像列表，设置工具栏按钮文本，启动工具栏的 EnableToolTips 方法激活提示功能。

```
BOOL CMyDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	    …                                    //代码省略部分参照 2.5 节主窗体设计
	m_Imagelist.Create(32,32,ILC_COLOR24|ILC_MASK,1,1);
```



```
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_login));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_open));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_add));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_pay));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_rishouru));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_reg));
    m_Imagelist.Add(AfxGetApp()->LoadIcon(IDI_ICON_cancel));
    UINT Array[7];
    for(int i=0;i<7;i++)
    {
        Array[i]=9000+i;
    }
    m_Toolbar.Create(this);
    m_Toolbar.SetButtons(Array,7);
    m_Toolbar.SetButtonText(0,"系统登录");
    m_Toolbar.SetButtonText(1,"开台");
    m_Toolbar.SetButtonText(2,"加减菜");
    m_Toolbar.SetButtonText(3,"顾客买单");
    m_Toolbar.SetButtonText(4,"本日收入");
    m_Toolbar.SetButtonText(5,"员工注册");
    m_Toolbar.SetButtonText(6,"退出系统");
    m_Toolbar.GetToolBarCtrl().SetButtonWidth(60,120);
    m_Toolbar.GetToolBarCtrl().SetImageList(&m_Imagelist);
    m_Toolbar.SetSizes(CSize(70,60),CSize(28,40));
    m_Toolbar.EnableToolTips(TRUE);                    //激活提示功能
    …                                                  //代码省略部分参照 2.5 节主窗体设计
    return true;
}
```

（2）在对话框的消息映射部分添加 ON_NOTIFY_EX 映射宏，如图 2.58 所示。

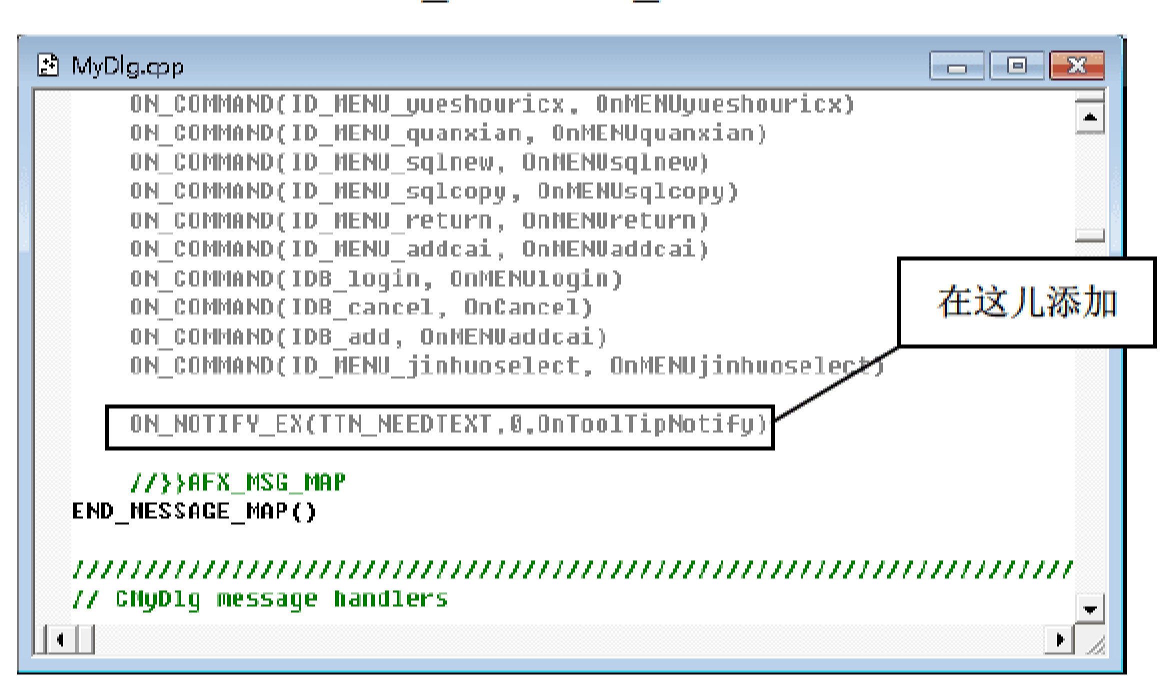
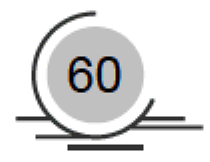


图 2.58　添加消息映射宏

（3）添加消息处理函数 OnToolTipNotify。声明和定义代码如下：

```
afx_msg BOOL OnToolTipNotify(UINT id, NMHDR* pNMHDR, LRESULT* pResult);
BOOL CMyDlg::OnToolTipNotify(UINT id, NMHDR *pNMHDR, LRESULT *pResult)
{
    TOOLTIPTEXT *pTTT = (TOOLTIPTEXT *)pNMHDR;
```



```
    UINT nID =pNMHDR->idFrom;                              //获取工具栏按钮 ID
    if(nID)
    {
        nID = m_Toolbar.CommandToIndex(nID);               //根据 ID 获取按钮索引
        if (nID != -1)
        {
            m_Toolbar.GetButtonText(nID,str);              //获取工具栏文本
            pTTT->lpszText = str.GetBuffer(str.GetLength()); //设置提示信息文本
            pTTT->hinst = AfxGetResourceHandle();
            return(TRUE);
        }
    }
    return(FALSE);
}
```

2.14　项目文件清单

餐饮管理系统的文件清单如表 2.8 所示。

表 2.8　餐饮管理设计清单

文件名	文件类型	说明	文件名	文件类型	说明
My.dsp	工程文件	工程文件	Jinhuoselect.cpp	源文件	进货查询窗体
My.dsw	工作区文件	工作区文件	Kaitaidlg.cpp	源文件	开台窗体
My.rc	资源文件	资源文件	Logindlg.cpp	源文件	登录窗体
My.cpp	源文件	工程头文件	Msado15.tlh	库文件	数据连接
MyDlg.cpp	源文件	主窗体头文件	Quanxiandlg.cpp	源文件	权限设置窗体
Copydlg.cpp	源文件	数据备份窗体文件	Rcxdlg.cpp	源文件	日收入查询窗体
CPdlg.cpp	源文件	菜式信息	Returndlg.cpp	源文件	数据库还原窗体
Diancaidlg.cpp	源文件	点菜窗体文件	SLdlg.cpp	源文件	点菜数量窗体
Jhselect.cpp	源文件	进货查询窗体	SpInfo.cpp	源文件	商品信息登记窗体
Jiacaidlg.cpp	源文件	加减菜窗体	Ycxdlg.cpp	源文件	月收入查询窗体
Jiezhangdlg.cpp	源文件	结账窗体	Zhucedlg.cpp	源文件	用户注册窗体
Jinhuodlg.cpp	源文件	进货信息窗体			

2.15　本 章 总 结

本章的主要内容是根据餐饮行业的实际情况设计一个管理系统。通过本章的学习，可以了解一个餐饮系统的开发流程，首先需要考虑系统的需求分析以及如何设计数据库，因为数据库设计直接影响了管理系统的好坏，任何一个好的管理系统的核心都是一个完善的数据库。本章通过详细的讲解以及简洁的代码使读者能够更快、更好地掌握数据库管理系统的开发技术。

第 3 章

客房管理系统

（Visual C++ 6.0+SQL Server 2014 实现）

随着市场经济的发展，人们生活水平的不断提高，以及到异地办公、旅游的人数的增多，宾馆酒店业不断壮大，人们对住宿的要求也不断提高。传统的手工管理已经不能适应复杂的客房管理需求，各宾馆为了提高管理水平都先后使用计算机进行管理，这就需要开发出符合客房管理要求的管理系统，本章以软件工程的思想介绍了客房管理系统的开发过程。

通过学习本章，读者可以学到：

- 使用 SQL Server 2014 数据库
- 使用 ADO 连接数据库
- 通过 SQL 语句对数据库进行操作


配置说明

3.1　开发背景

随着我国市场经济的迅速发展，人们的生活水平有了显著提高，旅游经济和各种商务活动更促进了酒店行业的快速发展。同时，随着宾馆、酒店的数量越来越多，人们的要求也越来越高，住宿行业的竞争愈演愈烈。如何在激烈的市场竞争中生存和发展，是每一个宾馆、酒店必须面临的问题。提高宾馆、酒店的经营管理，为顾客提供更优质的服务，同时降低运营成本是发展的关键。面对信息时代的机遇和挑战，利用科技手段提高企业管理效率无疑是一条行之有效的途径。计算机的智能化管理技术可以极大限度地提高服务管理水平，进行准确、快捷和高效的管理。因此，采用全新的计算机客房管理系统，已成为提高宾馆、酒店管理效率，改善服务水平的重要手段之一。管理方面的信息化已成为现代化管理的重要标志。

以往的人工操作管理中存在着许多问题，例如：

☑　人工计算账单容易出现错误。
☑　收银工作中容易发生账单丢失。
☑　客人具体消费信息难以查询。
☑　无法对以往营业数据进行查询。

3.2　需求分析

根据宾馆的具体情况，系统主要功能包括住宿管理、客房管理、挂账管理、查询统计、日结、系统设置。



视频讲解

3.3　系统设计

3.3.1　系统目标

面对酒店行业的高速发展和酒店行业信息化发展的过程中出现的各种情况，酒店客房管理系统应能够达到以下目标。

☑　实现多点操作的信息共享，相互之间的信息传递准确、快捷和顺畅。
☑　服务管理信息化，可随时掌握客人住宿、挂账率、客房状态等情况。
☑　系统界面友好美观，操作简单易行，查询灵活方便，数据存储安全。
☑　客户档案、挂账信息和预警系统相结合，可对往来客户进行住宿监控，防止坏账的发生。
☑　通过酒店客房管理系统的实施，可逐步提高酒店客房的管理水平，提升员工的素质。
☑　系统维护方便可靠，有较高的安全性，满足实用性、先进性的要求。

3.3.2　系统功能结构

根据该客房的具体情况，系统主要功能包括以下几个方面。

☑　住宿管理：客房预订、调房登记、入住登记、续住登记和退房登记。

☑　客房管理：房态设置、宿费提醒和房态查询。

☑　挂账管理：客房管理和客户结款。

☑　查询统计：住宿查询、退宿查询和宿费提醒。

☑　日结：登记预收报表、客房销售报表和客房销售统计。

☑　系统设置：初始化、密码设置和权限设置。

为了清晰、全面地介绍客房管理系统的功能，以及各个模块间的从属关系，下面以结构图的形式展现系统功能，如图 3.1 所示。

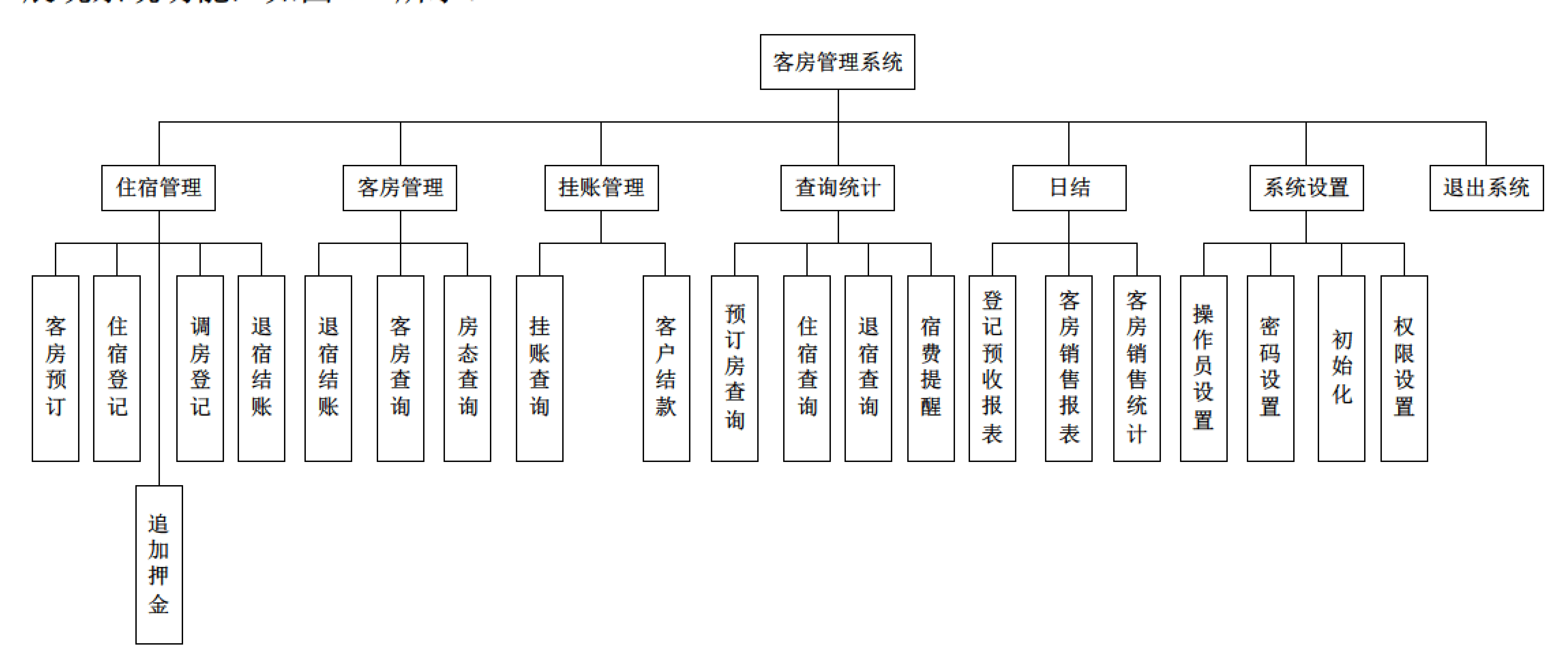


图 3.1　系统功能结构图

3.3.3　系统　览

本系统包含多个功能模块，这里给出主要的窗体界面图，帮助大家更快地了解本系统的结构功能。

主窗体包含打开其他窗体的菜单和主要功能的命令按钮，是程序最主要的界面。其运行效果如图 3.2 所示。

客房预订模块主要用来记录客户的预订客房信息，实现对预订信息的管理。其界面效果如图 3.3 所示。

追加押金模块主要用来实现记录追加押金的信息，并显示客人的当前住宿信息。其运行界面如图 3.4 所示。

调房登记模块主要用来实现记录客人调房信息。其运行界面如图 3.5 所示。

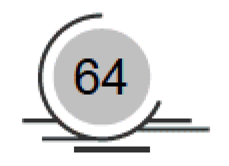




图 3.2　系统主界面



图 3.3　“客房预订”界面

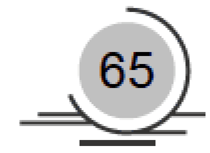




图 3.4　“追加押金”界面　　　　图 3.5　“调房登记”界面

3.3.4　业务流程图

客房管理系统业务流程图如图 3.6 所示。



顾客

预订房间　登记入住　调房　退房结账

前台

住宿管理　客房管理　查询统计　报表管理　挂账管理　系统设置

图 3.6　客房管理系统流程图

3.3.5　数据库设计

1．数据库概要说明

在 SQL Server 2014 数据库中建立名为 myhotel 的数据库，设计以下数据表：checkinregtable、checkoutregtable、guazhanginfo、kfyd、regmoneytable、roomsetting、setability 和 usertalbe。

如图 3.7 所示即为本系统数据库中的数据表结构图，该结构图中包含系统所有的数据表，可以清晰地反映数据库信息。



图 3.7　数据库概要说明

2．主要数据表结构

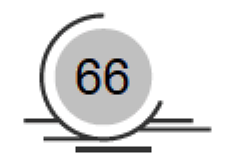
下面给出主要数据表的结构，其他表的结构参见数据库。

☑ 住宿登记表：主要用于记录住宿登记信息，包括住宿人信息、房间信息和住宿情况，该表结构如图 3.8 所示。

☑ 退宿登记表：主要用于记录退房登记信息，包括住宿和退房情况等信息，该表结构如图 3.9 所示。

☑ 客房设置表：用于存储客房的基本信息和客房状态等信息，该表结构如图 3.10 所示。

☑ 客房预订表：用于记录客房预订信息，包括预订人信息和房间信息等，该表结构如图 3.11 所示。





MRKJ_ZHD\EAST.my...o.checkinregtable

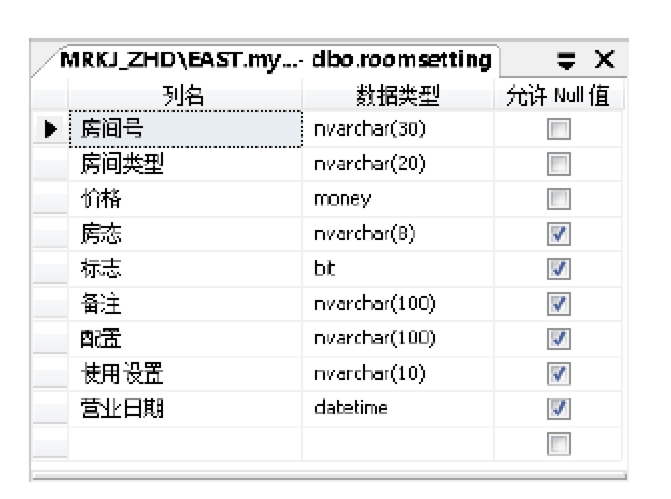
列名	数据类型	允许 Null 值
凭证号码	nvarchar(20)	☑
姓名	nvarchar(50)	☑
证件名称	nvarchar(20)	☑
证件号码	nvarchar(20)	☑
详细地址	nvarchar(50)	☑
出差事由	nvarchar(50)	☑
房间号	nvarchar(20)	☑
客房类型	nvarchar(10)	☑
联系电话	nvarchar(20)	☑
客房价格	money	☑
住宿日期	datetime	☑
住宿时间	datetime	☑
住宿天数	float	☑
宿费	money	☑
折扣	float	☑
应收宿费	money	☑
预收金额	money	☑
提醒日期	datetime	☑
退宿日期	datetime	☑
备注	nvarchar(50)	☑
标志	nvarchar(1)	☑
日期	datetime	☑
时间	datetime	☑
结款方式	nvarchar(10)	☑
退宿时间	datetime	☑
提醒时间	datetime	☑
摘要	nvarchar(200)	☑
BZ	float	☑
		☐

图 3.8　住宿登记表

MRKJ_ZHD\EAST.my...checkoutregtable

列名	数据类型	允许 Null 值
凭证号码	nvarchar(20)	☑
姓名	nvarchar(50)	☑
证件名称	nvarchar(20)	☑
证件号码	nvarchar(20)	☑
详细地址	nvarchar(50)	☑
工作单位	nvarchar(50)	☑
房间号	nvarchar(20)	☑
客房类型	nvarchar(10)	☑
客房价格	money	☑
住宿日期	datetime	☑
住宿时间	datetime	☑
住宿天数	float	☑
宿费	money	☑
折扣或招待	nvarchar(16)	☑
折扣	float	☑
应收宿费	money	☑
杂费	money	☑
电话费	money	☑
会议费	money	☑
存车费	money	☑
赔偿费	money	☑
金额总计	money	☑
预收宿费	money	☑
找还宿费	money	☑
退房日期	datetime	☑
退房时间	datetime	☑
日期	datetime	☑
时间	datetime	☑
备注	nvarchar(50)	☑
联系电话	nvarchar(20)	☑
BZ	float	☑

图 3.9　退宿登记表



MRKJ_ZHD\EAST.my...- dbo.roomsetting

列名	数据类型	允许 Null 值
房间号	nvarchar(30)	☐
房间类型	nvarchar(20)	☐
价格	money	☐
房态	nvarchar(8)	☑
标志	bit	☑
备注	nvarchar(100)	☑
配置	nvarchar(100)	☑
使用设置	nvarchar(10)	☑
营业日期	datetime	☑
		☐

图 3.10　客房设置表

MRKJ_ZHD\EAST.myhotel - dbo.kfyd

列名	数据类型	允许 Null 值
姓名	nvarchar(50)	☑
身份证号	nvarchar(20)	☑
联系电话	nvarchar(30)	☑
详细地址	nvarchar(100)	☑
工作单位	nvarchar(50)	☑
客房类型	nvarchar(10)	☑
房间价格	nvarchar(20)	☑
预住日期	datetime	☑
预住天数	nvarchar(10)	☑
预付金额	money	☑
备注	nvarchar(50)	☑
日期	datetime	☑
操作员	nvarchar(50)	☑
时间	datetime	☑
证件名称	nvarchar(20)	☑

图 3.11　客房预订表

3.4　主窗体设计



视频讲解

3.4.1　主窗体概

主程序界面是应用程序提供给用户访问其他功能模块的平台，根据实际需要，客房管理系统的主界面采用了传统的“菜单/工具栏/状态栏”风格。客房管理系统的主程序界面如图 3.2 所示。

3.4.2　主窗体实现　程

1．客户区设计

在生成的对话框内添加图片、静态文本、标签、编辑框和按钮等资源。

控件的属性和 ID 如表 3.1 所示。

表 3.1　控件的属性和 ID

控件 ID	标	控件 ID	标
ID_BTN_borrowroom	开房	ID_BTN_daysummery	日结
ID_BTN_returnroom	结账	ID_BTN_alert	提醒
ID_BTN_mainfind	查询	ID_CLOSE	退出

2. 菜单设计

（1）选择 Insert→Resource 命令，打开 Insert Resource 对话框，如图 3.12 所示。

（2）选择 Menu 选项，单击 New 按钮，插入空白菜单，设置 ID 属性为 IDR_mainMENU，然后按照如图 3.13 所示的界面编辑菜单项。

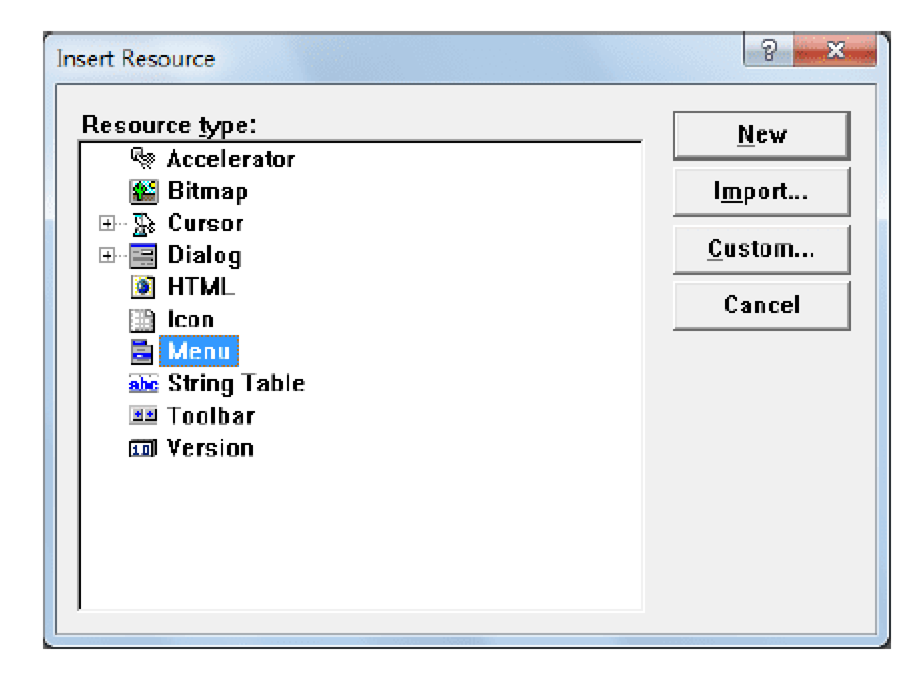


图 3.12　Insert Resource 对话框

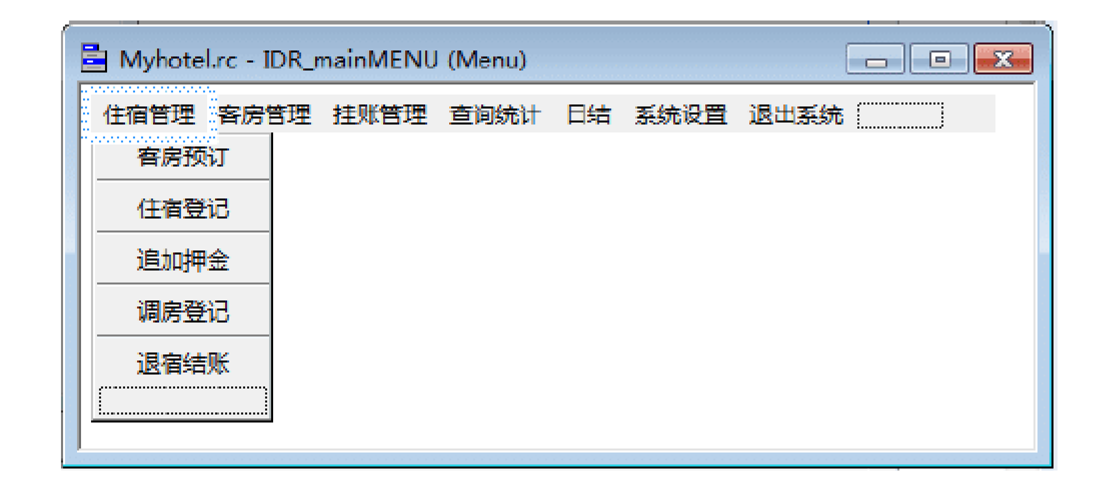


图 3.13　菜单资源

主菜单的各个子菜单的 ID 和标题属性如表 3.2 所示。

表 3.2　各个子菜单的 ID 和标　属性

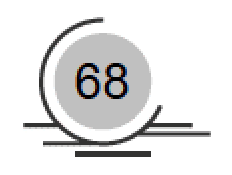
控件 ID	标	控件 ID	标
ID_MENU_checkinreg	住宿登记	ID_MENU_regmoneytable	登记预收报表
ID_MENU_roomsetting	客房设置	ID_MENU_saleroomtable	客房销售报表
ID_MENU_checkout	退宿结账	ID_MENU_saleroomsummary	客房销售统计
ID_MENU_addmoney	追加押金	ID_MENU_adm_setting	操作员设置
ID_MENU_changeroomreg	调房登记	ID_MENU_pwd_setting	密码设置
ID_MENU_findroom	客房查询	ID_MENU_setting_begin	初始化
ID_MENU_findguazhang	挂账查询	ID_MENU_setting_ability	权限设置
ID_MENU_guazhangmoney	客户结款	ID_MENU_findroomstate	房态查看
ID_MENU_findcheckinreg	住宿查询	ID_MENU_roomprebook	客房预订
ID_MENU_findcheckoutreg	退宿查询	ID_MENU_findprebookroom	预订房查询
ID_MENU_findroomfee	宿费提醒		

3. 代码分析

（1）系统主界面操作可以根据用户的权限设定，所以应加入连接数据库功能，故在 stdafx.h 文件中加入以下代码，提供加入 ADO 的支持。

```
//添加 ADO 支持
#import "c:\program files\common files\system\ado\msado15.dll" \ no_namespace \ rename ("EOF", "adoEOF")
```

并在 Myhotel.h 中加入以下代码：





```
  CDatabase m_DB;
_ConnectionPtr m_pConnection;
```

此外，在 myhotel.cpp 的初始化函数中加入连接数据库的代码：

```
try                                              //   接数据库
{
 CString strConnect;
 strConnect.Format("DSN=myhotel;");
 if(!m_DB.OpenEx(strConnect,CDatabase::useCursorLib))
 {
 AfxMessageBox("Unable to Connect to the Specified Data Source");
 return FALSE;
 }
}
catch(CDBException *pE)                          //抛出异常
{
    pE->ReportError();
    pE->Delete();
    return FALSE;
}
    //初始化 COM，创建 ADO   接等操作
    AfxOleInit();
    m_pConnection.CreateInstance(__uuidof(Connection));
    //在 ADO 操作中建议语句中要常用 try...catch()来捕获   误信息
    try
    {
        //打开本地数据库
m_pConnection->Open("Provider=MSDASQL.1;Persist  Security  Info=False;Data  Source  =  myhotel","","",
adModeUnknown);
    }
    catch(_com_error e)                          //抛出可能发生的异常
    {
        AfxMessageBox("数据库   接失败，确认数据库   置正确!");
        return FALSE;
    }
```

（2）主窗口初始化时，需要根据登录操作员的权限来设置其可以进行的操作，此功能由函数 setuserability()来完成，代码如下：

```
void CMyhotelDlg::setuserability()
{
    m_pRecordset.CreateInstance(__uuidof(Recordset));
    _variant_t var,varIndex;

    //loguserid="操作员 01";
    CString strsqlshow;
    strsqlshow.Format("SELECT * FROM setability where 操作员='%s'",loguserid);
```

```
        try                                                     //打开数据库   接
        {
            m_pRecordset->Open((_variant_t)(strsqlshow),        //查询表中所有字段
                theApp.m_pConnection.GetInterfacePtr(),         //获取   接库的 IDispatch 指
                                adOpenDynamic,
                                adLockOptimistic,
                                adCmdText);
        }
        catch(_com_error *e)                                    //捕获异常的发生
        {
            AfxMessageBox(e->ErrorMessage());
        }
❶       mynenu=AfxGetMainWnd()->GetMenu();                      //获得主菜单指
        CString ling="0";
    }
    try
        {
            if(!m_pRecordset->BOF)                              //判断指   是否在数据   最后
                m_pRecordset->MoveFirst();
            else
            {
                AfxMessageBox("表内数据为空");
                return;
            }
             MessageBox("eeeeeeeeee");
             //读取数据表内客房   订字段内容
             var = m_pRecordset->GetCollect("客房   订");
             if(var.vt != VT_NULL)
             {
               if((LPCSTR)_bstr_t(var)==ling)                   //判断是否有权   操作客房   订模块
               {                                                //如果没有权   就使该菜单呈灰色显示
               EnableMenuItem(mynenu->m_hMenu,ID_MENU_roomprebook,MF_DISABLED|MF_GRAYED);
               }

            }
            //读取数据表内住宿登记字段内容
            var = m_pRecordset->GetCollect("住宿登记");
            if(var.vt != VT_NULL)
            {
               if((LPCSTR)_bstr_t(var)==ling)                   //判断是否有权   操作住宿登记模块
               {                                                //如果没有权利就使该菜单呈灰色显示
               EnableMenuItem(mynenu->m_hMenu,ID_MENU_checkinreg,MF_DISABLED|MF_GRAYED);
               }

            }
            //读取数据表内   加押   字段内容
            var = m_pRecordset->GetCollect("   加押   ");
            if(var.vt != VT_NULL)
            {
```

```
            if((LPCSTR)_bstr_t(var)==ling)                  //判断是否有权  操作  加押  模块
            {                                               //如果没有权利就使该菜单呈灰色显示
            EnableMenuItem(mynenu->m_hMenu,ID_MENU_addmoney,MF_DISABLED|MF_GRAYED);
            }

        }
        //读取数据表内调房登记字段内容
        var = m_pRecordset->GetCollect("调房登记");
        if(var.vt != VT_NULL)
        {
            if((LPCSTR)_bstr_t(var)==ling)                  //判断是否有权  操作调房登记模块
             {                                              //如果没有权利就使该菜单呈灰色显示
             EnableMenuItem(mynenu->m_hMenu,ID_MENU_changeroomreg,MF_DISABLED |MF_GRAYED);
             }

        }

        //其他菜单设计代码参见资源包
            mynenu->Detach();
        DrawMenuBar();                                      //  绘主菜单
    catch(_com_error *e)                                    //捕获异常
    {
        AfxMessageBox(e->ErrorMessage());                   //弹出  误信息框
    }

    m_pRecordset->Close();                                  //关  记录
    m_pRecordset = NULL;

}
```

代码贴士

❶ GetMenu 函数：获得窗口的菜单指针，能对子窗口使用，因为它们没有菜单。返回的指针可能是临时的，不能被保存以供将来使用。

（3）在实现主窗体时，需要创建几个函数，创建 OnSysCommand 函数的代码如下：

```
void CMyhotelDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
     if ((nID & 0xFFF0) == IDM_ABOUTBOX)
     {
          CAboutDlg dlgAbout;
          dlgAbout.DoModal();
     }
     else
     {
          CDialog::OnSysCommand(nID, lParam);
     }
}
```

创建 OnPaint()函数，代码如下：

```
void CMyhotelDlg::OnPaint()
{CPaintDC dc(this);
	CBitmap bit;
	CDC memDC;
	CRect rect;
	this->GetClientRect(&rect);

	bit.LoadBitmap(IDB_MAINBK);

	BITMAP bmpInfo;
	bit.GetBitmap(&bmpInfo);
	int imgWidth = bmpInfo.bmWidth;
	int imgHeight = bmpInfo.bmHeight;
	memDC.CreateCompatibleDC(&dc);
	memDC.SelectObject(&bit);
	dc.StretchBlt(0,0,rect.Width(),rect.Height(),&memDC,0,0,imgWidth,imgHeight,SRCCOPY);
	memDC.DeleteDC();
	bit.DeleteObject();

}
```

创建 OnQueryDragIcon()、OnMENUcheckinreg()、OnBTNborrowroom()函数，代码如下：

```
HCURSOR CMyhotelDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CMyhotelDlg::OnMENUcheckinreg()
{
	CCheckinregdlg mycheckindlg;
      mycheckindlg.DoModal();
}

void CMyhotelDlg::OnBTNborrowroom()
{
	OnMENUcheckinreg();
}
```

创建 OnMENUroomsetting()、OnMENUcheckout()、OnBTNreturnroom()函数，代码如下：

```
void CMyhotelDlg::OnMENUroomsetting()
{
	CSetroomdlg mysetroomdlg;
	mysetroomdlg.DoModal();
```



```
}

void CMyhotelDlg::OnMENUcheckout()
{
     CCheckoutdlg mycheckoutdlg;
    mycheckoutdlg.DoModal();
}

void CMyhotelDlg::OnBTNreturnroom()
{
     OnMENUcheckout();
}
```

创建 OnMENUaddmoney()、OnMENUchangeroomreg()、OnMENUfindroom()函数，代码如下：

```
void CMyhotelDlg::OnMENUaddmoney()
{
     CAddmoneydlg myaddmoneydlg;
    myaddmoneydlg.DoModal();
}

void CMyhotelDlg::OnMENUchangeroomreg()
{
     CChangeroomdlg mychangeroomdlg;
    mychangeroomdlg.DoModal();
}

void CMyhotelDlg::OnMENUfindroom()
{
     CFindroomdlg myfindroomdlg;
    myfindroomdlg.DoModal();
}
```

3.5　登录模块设计


视频讲解

3.5.1　登录模块概

为了防止非法用户进入系统，本软件设计了系统登录窗口。在程序启动时，首先弹出“登录”窗口，要求用户输入登录信息，如果用户输入不合法，将禁止进入系统。登录模块的运行效果如图 3.14 所示。



图 3.14　登录模块的运行效果

3.5.2　登录模块技术分析

本模块使用 CUserset 类实现对数据源的连接。这里是通过 ODBC 数据源进行连接的，在连接数据库之前，要先在系统上创建一个名为 myhotel 的数据源。userset.cpp 中的代码如下：

```
CString CUserset::GetDefaultConnect()
{
	return _T("ODBC;DSN=myhotel");
}
CString CUserset::GetDefaultSQL()
{
	return _T("[dbo].[user]");
}
```

3.5.3　登录模块实现　程

本模块使用的数据表：usertalbe

（1）选择 Insert→Resource 命令，打开添加资源界面。选择 Dialog 选项，单击 New 按钮，插入新的对话框。

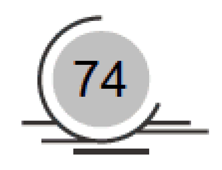
（2）利用类向导为此对话框资源设置属性。在 Name 文本框中输入对话框类名，如 CLoginDlg，在 Base class 下拉列表框中选择一个基类，这里为 CDialog，单击 OK 按钮创建对话框。

（3）在工作区的资源视图中选择新创建的对话框，向对话框中添加静态文本、下拉列表框、编辑框和按钮等资源。主要资源属性如表 3.3 所示。

表 3.3　主要资源属性

控件 ID	对应变　/标　属性	控件 ID	对应变　/标　属性
IDC_COMBO_username	m_username	IDOK	确定
IDC_password	m_password	IDCANCEL	取消

（4）建立和数据库的映射，利用类向导建立记录集的映射类，如图 3.15 所示。





选择基类为 CDaoRecordset，单击 OK 按钮进入下一步，如图 3.16 所示。

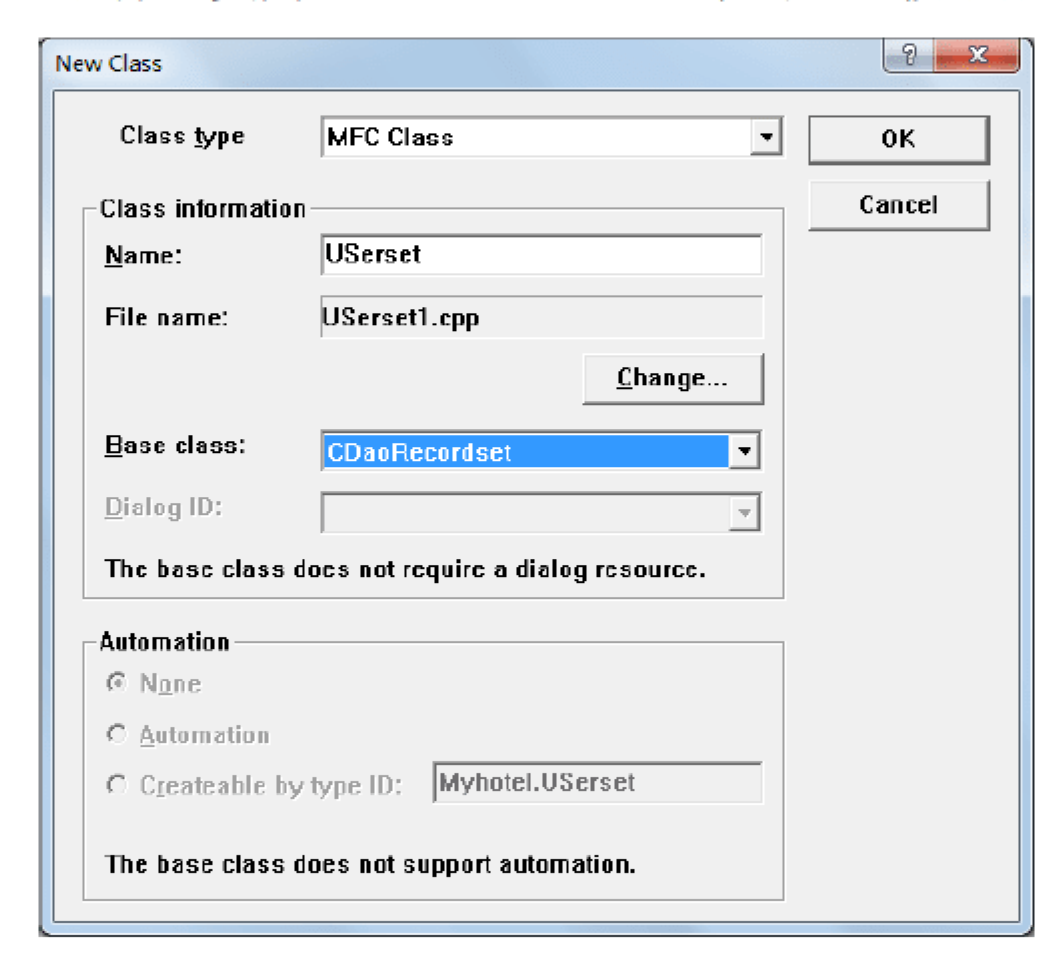


图 3.15　New Class 对话框

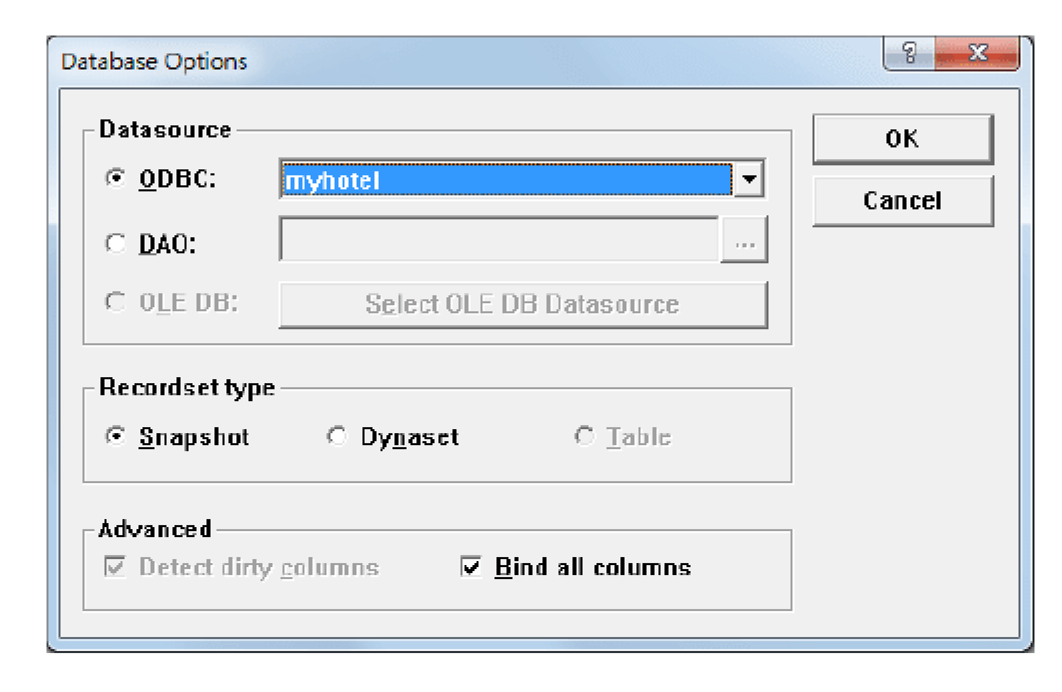


图 3.16　Database Options 对话框

选择数据源类型为 ODBC，并选择所使用的数据源，此处选择 myhotel 数据源，单击 OK 按钮，进入下一步，如图 3.17 所示。

选择所要关联的数据表，因为是操作员登录信息，所以选择 dbo.usertable 数据表，单击 OK 按钮完成映射。

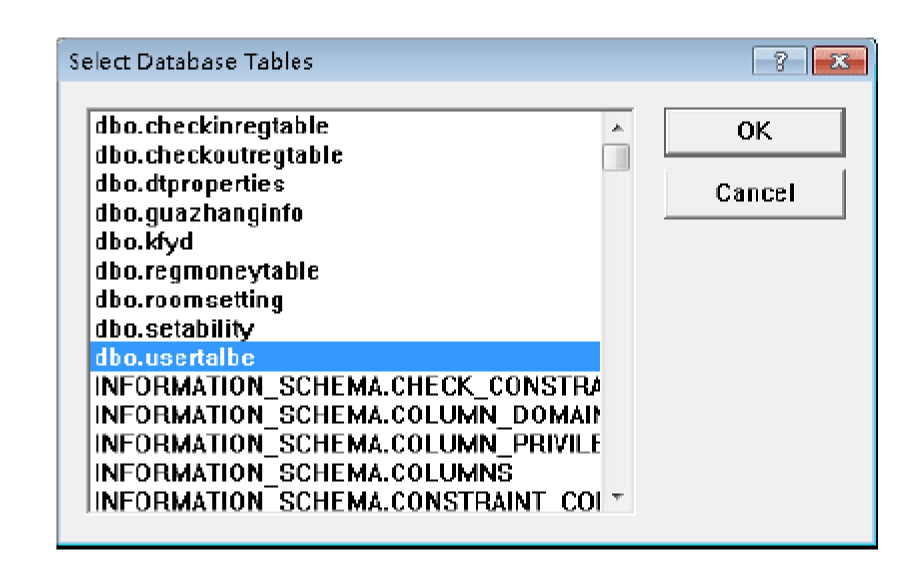


图 3.17　Select Database Tables 对话框

可以看到已经创建了一个新类 CUserset，其头文件的关键代码如下：

```
class CUserset : public CRecordset
{
public:
	CUserset(CDatabase* pDatabase = NULL);
	DECLARE_DYNAMIC(CUserset)

	//{{AFX_FIELD(CUserset, CRecordset)
	CString m_user_name;
	CString m_user_pwd;
	//}}AFX_FIELD
	//{{AFX_VIRTUAL(CUserset)
	public:
	virtual CString GetDefaultConnect();
	virtual CString GetDefaultSQL();
	virtual void DoFieldExchange(CFieldExchange* pFX);
	//}}AFX_VIRTUAL

#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
```

```
#endif
};
```

（5）单击“确定”按钮可以登录到系统主界面，此按钮的相应函数如下：

```
void CLoginDlg::OnOK()
{
    CString sqlStr;
    UpdateData(true);
    if(m_username.IsEmpty())                                    //判断用户名是否为空
    {
        AfxMessageBox("请  入用户名!");
        return;
    }
    //创建查询语句
    sqlStr="SELECT * FROM usertalbe WHERE user_name='";
    sqlStr+=m_username;
    sqlStr+="'";
    sqlStr+="AND user_pwd='";
    sqlStr+=m_password;
    sqlStr+="'";
    //打开数据库
    if(!myuserset.Open(AFX_DB_USE_DEFAULT_TYPE,sqlStr))
    {
        AfxMessageBox("user 表打开失败!");
        return;
    }
    loguserid=m_username;                                       //保存操作员 ID，其他窗口中会用到该数据

    if(!myuserset.IsEOF())                                      //关  数据库  接
    {
        myuserset.Close();
        CDialog::OnOK();
    }
    else
    {                                                           //给出  误提示
        AfxMessageBox("登录失败!");
        m_username=_T("");
        m_password=_T("");
        UpdateData(false);                                      //更新显示
        myuserset.Close();                                      //关  数据库  接
        return;
    }

}
```

（6）为使按下 Enter 键时控制输入焦点，故加入 PreTranslateMessage 方法，代码如下：

```
BOOL CLoginDlg::PreTranslateMessage(MSG* pMsg)
{
```

```
    if(pMsg->message==WM_KEYDOWN&&pMsg->wParam==VK_RETURN)
    {

        DWORD def_id=GetDefID();
         if(def_id!=0)
         {
             //MSG 消息的结构中的 hwnd 存储的是接收该消息的窗口句柄
             CWnd *wnd=FromHandle(pMsg->hwnd);
                 char class_name[16];
             if(GetClassName(wnd->GetSafeHwnd(),class_name,sizeof(class_name))!=0)
             {
                 DWORD style=::GetWindowLong(pMsg->hwnd,GWL_STYLE);
                 if((style&ES_MULTILINE)==0)
                 {
                     if(strnicmp(class_name,"edit",5)==0)
                     {   //将焦点设置到   认按   上
                         GetDlgItem(LOWORD(def_id))->SetFocus();
                   pMsg->wParam=VK_TAB;                          //      Enter   消息为 Tab   消息
                      }
                 }
             }
         }
    }

    return CDialog::PreTranslateMessage(pMsg);
}
```

（7）登录模块与数据库连接代码如下：

```
BOOL CLoginDlg::OnInitDialog()
{
     CDialog::OnInitDialog();

               //使用 ADO 创建数据库记录
     m_pRecordset.CreateInstance(__uuidof(Recordset));

     _variant_t var;
     CString struser;
     //在 ADO 操作中建议语句中要常用 try...catch()来捕获   误信息
     try
     {//打开数据库
          m_pRecordset->Open("SELECT * FROM usertalbe",                //查询表中所有字段
                             theApp.m_pConnection.GetInterfacePtr(), //获取   接库的 IDispatch 指
                             adOpenDynamic,
                             adLockOptimistic,
                             adCmdText);
     }
     catch(_com_error *e)//捕获打开数据库可能发生的异常情况并实时显示提示
```

```
	{
		AfxMessageBox(e->ErrorMessage());
	}
	try
	{
		if(!m_pRecordset->BOF)                    //判断指　是否在数据　最后
			m_pRecordset->MoveFirst();
//		else
//		{//提示　误，无数据
//			AfxMessageBox("表内数据为空");
//			return false;
//		}

		while(!m_pRecordset->adoEOF)
		{
			var = m_pRecordset->GetCollect("user_name");
			if(var.vt != VT_NULL)
				struser = (LPCSTR)_bstr_t(var);
			m_usernamectr.AddString(struser);  //从数据库获得的内容给变　赋值
			m_pRecordset->MoveNext();          //移动数据指
		}

	}
	catch(_com_error *e)                          //捕获异常
	{
		AfxMessageBox(e->ErrorMessage());
	}

	//关　记录
	m_pRecordset->Close();
	m_pRecordset = NULL;
	//更新显示
//	UpdateData(false);
	return TRUE;
}
```

（8）在登录界面中，需要对图片有限制，在 LoginDlg.cpp 文件中，写入如下代码：

```
void CLoginDlg::OnPaint()
{
	CPaintDC dc(this);
	CBitmap bit;
	CDC memDC;
	CRect rect;
	this->GetClientRect(&rect);

	bit.LoadBitmap(IDB_LOGINBK);
```

```
    BITMAP bmpInfo;
    bit.GetBitmap(&bmpInfo);
    int imgWidth = bmpInfo.bmWidth;
    int imgHeight = bmpInfo.bmHeight;
    memDC.CreateCompatibleDC(&dc);
    memDC.SelectObject(&bit);
    dc.StretchBlt(0,0,rect.Width(),rect.Height(),&memDC,0,0,imgWidth,imgHeight,SRCCOPY);
    memDC.DeleteDC();
    bit.DeleteObject();

}
```

3.6　客房预订模块设计



3.6.1　客房　订模块概

住宿管理模块包括客房预订、住宿登记、追加押金、调房登记、退宿结账等功能子模块。下面详细介绍客房预订子模块的设计。客房预订模块用于实现客房预订的功能，主要登记用户的姓名、证件、证件号码和预住日期等信息，是为预订客户提供服务的模块。其运行界面如图 3.3 所示。

3.6.2　客房　订模块技术分析

客房预订模块实现将预订客房信息插入到数据表中，主要是通过打开记录集，然后使用 AddNew 方法向数据表中插入一个新记录来实现对客房预订信息的添加。AddNew 方法用于向记录集中添加一个空行，然后设置这个空行的每个字段值，从而能够实现将一条记录添加到数据表中。

3.6.3　客房　订模块实现　程

本模块使用的数据表：kfyd

（1）选择 Insert→Resource 命令打开添加资源界面，选择 Dialog 选项，单击 New 按钮，插入新的对话框。

（2）利用类向导为此对话框资源设置属性。在 Name 文本框中输入对话框类名，如 CRoomprebookdlg，在 Base class 下拉列表框中选择一个基类，这里为 CDialog，单击 OK 按钮创建对话框。

（3）在工作区的资源视图中选择新创建的对话框，向对话框中添加静态文本、下拉列表框、编辑框、按钮和日期选择控件等资源。

各个主要控件的 ID 和属性设置如表 3.4 所示。

表 3.4　主要控件的 ID 和属性设置

控件 ID	变	控件 ID	变
IDC_COMBOprebookidkind	m_prebookidkind	IDC_prebookidnumber	m_prebookidnumber
IDC_COMBOroomkind	m_prebookroomkind	IDC_prebookname	m_prebookname
IDC_DATETIMEPICKERprecheckindate	m_prebookcheckindate	IDC_prebooktelnumber	m_prebooktelnumber
IDC_prebookaddr	m_prebookaddr	IDC_prebookworkcompany	m_prebookworkcompany
IDC_prebookdays	m_prebookdays	IDC_roommoney	m_prebookroommoney
IDC_prebookhandinmoney	m_prebookhandinmoney	IDC_STATICshowuser	m_showuser

（4）在其对应的头文件 Roomprebookdlg.h 中添加以下声明代码：

```
…
…
	CString gustname;
	CString gustaddr;

	CString zhengjian;
	CString zhengjian_number;
	CString checkinreg_reason;

	_ConnectionPtr m_pConnection;
	_CommandPtr m_pCommand;
	_RecordsetPtr m_pRecordset;
```

如果确定预订客房，单击“确定”按钮向数据库中插入预订记录，其响应函数如下：

```
void CRoomprebookdlg::OnOK()
{
		UpdateData(true);
	/*
	 *  检查  份证的号码是否为 15 位或者为 18 位
	 */
	CString strCertifyCode;                                  //证件号码
	//获得证件号码
	int nCertifyCodeLength=m_prebookidnumber.GetLength();    //获得证件号码的  度
	if(nCertifyCodeLength!=15&&nCertifyCodeLength!=18)
	{
		if(m_prebookidkind=="  份证")
		{                                                    //若  择的是  份证
			MessageBox("你的  份证的号码的位数不正确!\n 应该为 15 位或者 18 位!",
				"  份证  误",MB_OK);
			return;
		}
	}

	m_pRecordset.CreateInstance(__uuidof(Recordset));
	//在 ADO 操作中建议语句中要常用 try...catch()来捕获  误信息
```

```
try
{                                                           //打开数据表
    m_pRecordset->Open("SELECT * FROM kfyd",                //查询表中所有字段
    theApp.m_pConnection.GetInterfacePtr(),                 //获取　接库的 IDispatch 指
                    adOpenDynamic,
                    adLockOptimistic,
                    adCmdText);
}
catch(_com_error *e)                                        //捕获异常情况
{
    AfxMessageBox(e->ErrorMessage());
}

  try
  {
    //写入各字段值
    m_pRecordset->AddNew();
    //向数据表“姓名”字段写入数据
    m_pRecordset->PutCollect("姓名",_variant_t(m_prebookname));
    //向数据表“　份证号”字段写入数据
    m_pRecordset->PutCollect("　份证号", _variant_t(m_prebookidnumber));
    //向数据表“联系电话”字段写入数据
    m_pRecordset->PutCollect("联系电话", _variant_t(m_prebooktelnumber));
    //向数据表“详细地址”字段写入数据
    m_pRecordset->PutCollect("详细地址", _variant_t(m_prebookaddr));
    //向数据表“工作单位”字段写入数据
    m_pRecordset->PutCollect("工作单位", _variant_t(m_prebookworkcompany));
    //向数据表“客房类型”字段写入数据
    m_pRecordset->PutCollect("客房类型", _variant_t(m_prebookroomkind));
    //向数据表“客房价格”字段写入数据
    m_pRecordset->PutCollect("客房价格", _variant_t(m_prebookroommoney));
    CString checkindate;
    int nYear,nDay,nMonth;
    int nhour,nmin,nsecond;
    CString sYear,sDay,sMonth;
    nYear=m_prebookcheckindate.GetYear();                   //提取年
    nDay=m_prebookcheckindate.GetDay();                     //提取日
    nMonth=m_prebookcheckindate.GetMonth();                 //提取月
    sYear.Format("%d",nYear);                               //　换为字符串
    sDay.Format("%d",nDay);                                 //　换为字符串
    sMonth.Format("%d",nMonth);                             //　换为字符串
    //格式化时
    checkindate.Format("%s-%s-%s",sYear,sMonth,sDay);
    //向数据表“　住日期”字段写入数据
    m_pRecordset->PutCollect("　住日期",_variant_t(checkindate));
    //向数据表“　住天数”字段写入数据

    m_pRecordset->PutCollect("　住天数", _variant_t(m_prebookdays));
    //向数据表“　付　　”字段写入数据
```

```
        m_pRecordset->PutCollect("  付    ", _variant_t(m_prebookhandinmoney));
        CString nowdate,nowtime;
        CTime tTime;
❶       tTime=tTime.GetCurrentTime();
❷       nYear=tTime.GetYear();                          //提取年
        nDay=tTime.GetDay();                            //提取日
        nMonth=tTime.GetMonth();                        //提取月
        sYear.Format("%d",nYear);                       //  换为字符串
        sDay.Format("%d",nDay);                         //  换为字符串
        sMonth.Format("%d",nMonth);                     //  换为字符串
        //格式化时
        nowdate.Format("%s-%s-%s",sYear,sMonth,sDay);
        CString shour,smin,ssecond;
        nhour=tTime.GetHour();                          //提取小时
        nmin=tTime.GetMinute();                         //提取分
        nsecond=tTime.GetSecond();                      //提取秒
        shour.Format("%d",nhour);                       //  换为字符串
        smin.Format("%d",nmin);                         //  换为字符串
        ssecond.Format("%d",nsecond);                   //  换为字符串
        //格式化时
        nowtime.Format("%s:%s:%s",shour,smin,ssecond);
        m_pRecordset->PutCollect("日期", _variant_t(nowdate));
        m_pRecordset->PutCollect("时  ", _variant_t(nowtime));
        //向数据表“证件名称”字段写入数据
        m_pRecordset->PutCollect("证件名称", _variant_t(m_prebookidkind));
        //更新数据表
        m_pRecordset->Update();
        AfxMessageBox("  订成功!");
       }

        catch(_com_error *e)//抛出异常情况，并显示
        {
        AfxMessageBox(e->ErrorMessage());
        }

 //关  记录
        m_pRecordset->Close();
m_pRecordset = NULL;
}
```

代码贴士

❶ GetCurrentTime 函数：此成员函数返回一个代表当前时间的 CTime 对象。

❷ GetYear 函数：此成员函数根据本地时间返回范围在 1970 年 1 月 1 日～2038 年 1 月 18 日之间的年份。

在预定房间时，需要选择一个房间的类型，实现的具体代码如下：

```
void CRoomprebookdlg::OnCloseupCOMBOroomkind()
{
    CString roomkind;
```



```
    //获得　入值
    UpdateData(true);
    roomkind=m_prebookroomkind;
    //如果客房类型是标房
    if(m_prebookroomkind=="标房")
    {
        m_prebookroommoney="138";

    }
    //如果客房类型是普房
    if(m_prebookroomkind=="普房")
    {
        m_prebookroommoney="98";

    }
    //如果客房类型是双人
    if(m_prebookroomkind=="双人　")
    {
        m_prebookroommoney="168";

    }
    //如果客房类型是套房
    if(m_prebookroomkind=="套房")
    {
        m_prebookroommoney="268";

    }
    //更新显示
    UpdateData(false);

}
```

如果顾客想在入住之前换房间，就需要重新修改预订信息，实现这个功能的代码如下：

```
void CRoomprebookdlg::Oncancelprebookroom()
{
    //　入变　初始化
    m_prebookidkind = _T("");
    m_prebookroomkind = _T("");
    m_prebookcheckindate = 0;
    m_prebookaddr = _T("");
    m_prebookdays = _T("");
    m_prebookhandinmoney = _T("");
    m_prebookidnumber = _T("");
    m_prebookname = _T("");
    m_prebooktelnumber = _T("");
    m_prebookworkcompany = _T("");
    m_prebookroommoney = _T("");
```

```
	CTime tTime;
	tTime=tTime.GetCurrentTime();
	//设置登记的  认时
	m_prebookcheckindate=tTime;
	//更新显示
	UpdateData(false);

}

BOOL CRoomprebookdlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	m_showuser=loguserid;
	enable(0);
	//更新  入框状态

	CTime tTime;
	tTime=tTime.GetCurrentTime();
	//设置登记的  认时
	m_prebookcheckindate=tTime;
	UpdateData(false);

	return TRUE;
}

void CRoomprebookdlg::OnBtnroomyuding()
{
	enable(1);
	//更新  入框状态
}
void CRoomprebookdlg::enable(bool bEnabled)
{
	//m_ComYSFS.EnableWindow(bEnabled);
	//更改  入框等控件状态，方便使用，  止  误操作
	GetDlgItem(IDC_COMBOprebookidkind)->EnableWindow(bEnabled);
	GetDlgItem(IDC_COMBOroomkind)->EnableWindow(bEnabled);
	GetDlgItem(IDC_DATETIMEPICKERprecheckindate)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebookaddr)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebookdays)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebookhandinmoney)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebookidnumber)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebookname)->EnableWindow(bEnabled);
	GetDlgItem(IDC_prebooktelnumber)->EnableWindow(bEnabled);

	GetDlgItem(IDC_prebookworkcompany)->EnableWindow(bEnabled);
	GetDlgItem(IDC_roommoney)->EnableWindow(bEnabled);
	GetDlgItem(IDOK)->EnableWindow(bEnabled);
```

```
        GetDlgItem(IDcancelprebookroom)->EnableWindow(bEnabled);

}
```

3.7　追加押金模块设计



3.7.1　加押　模块概

追加押金是为方便客户追加预交的住房押金而设计的，在此子对话框中只要选择客户的凭证号码，然后输入追加的金额就可以轻松地完成追加操作，其运行界面如图 3.4 所示。

3.7.2　加押　模块技术分析

追加押金模块用于将追加押金信息记录到数据表中。在打开窗体时，“凭证号码”下拉列表框中自动显示了当前数据库中的凭证号码，可直接在此选择一个凭证号码。这个凭证号码是在窗体初始化时添加到下拉列表框中的。通过查询符合条件的记录，使用循环语句将记录添加到组合框中，其实现代码如下：

```
while(!m_pRecordset->adoEOF)
{
        var = m_pRecordset->GetCollect("凭证号码");
        if(var.vt != VT_NULL)
                strregnumber = (LPCSTR)_bstr_t(var);
        m_addmoney_regnumberctr.AddString(strregnumber);
        m_pRecordset->MoveNext();                          //移动记录指　到下一条记录
}
```

3.7.3　加押　模块实现　程

本模块使用的数据表：checkinregtable

（1）选择 Insert→Resource 命令，打开添加资源界面，选择 Dialog 选项，单击 New 按钮，插入新的对话框。

（2）利用类向导为此对话框资源设置属性。在 Name 文本框中输入对话框类名，如 CAddmoneydlg，在 Base class 下拉列表框中选择一个基类，这里为 CDialog，单击 OK 按钮创建对话框。

（3）在工作区的资源视图中选择新创建的对话框，向对话框中添加静态文本、下拉列表框、编辑框、按钮和时间日期选择控件等资源。

各个控件的 ID 和属性设置如表 3.5 所示。

表 3.5　各控件的 ID 和属性设置

控件 ID	对 应 变	控件 ID	对 应 变
IDC_COMBO_regnumber	m_addmoney_regnumberctr	IDC_EDIT_roomnumber	m_addmoney_roomnumber
IDC_COMBO_regnumber	m_addmoney_regnumber	IDC_EDIT_alarmdate	m_addmoney_alarmdate
IDC_EDIT_name	m_addmoney_name	IDC_EDIT_alarmtime	m_addmoney_alarmtime
IDC_EDIT_outdate	m_addmoney_outdate	IDC_EDIT_checkdays	m_addmoney_checkdays
IDC_EDIT_outtime	m_addmoney_outtime	IDC_EDIT_indate	m_addmoney_indate
IDC_EDIT_prehandmoney	m_addmoney_prehandmoney	IDC_EDIT_intime	m_addmoney_intime
IDC_EDIT_roomlevel	m_addmoney_roomlevel	IDC_addmoney	m_addmoney
IDC_EDIT_roommoney	m_addmoney_roommoney	IDC_STATICshowuser	m_showuser

（4）在对应类的头文件 Addmoneydlg.h 中声明以下变量：

```
_ConnectionPtr m_pConnection;
_CommandPtr m_pCommand;
_RecordsetPtr m_pRecordset;
_RecordsetPtr m_pRecordsetout;
```

对话框的初始化函数完成住宿客户凭证号码的准备等其他的初始化工作，该对话框类的初始化函数如下：

```
BOOL CAddmoneydlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //使用 ADO 创建数据库记录
    m_pRecordset.CreateInstance(__uuidof(Recordset));

        _variant_t var;
        CString strregnumber;
    //在 ADO 操作中建议语句中要常用 try...catch()来捕获　误信息
    try
    {
m_pRecordset->Open("SELECT * FROM checkinregtable",          //查询表中所有字段
    theApp.m_pConnection.GetInterfacePtr(),                  //获取　接库的 IDispatch 指
    adOpenDynamic,
    adLockOptimistic,
    adCmdText);
    }
    catch(_com_error *e)                                     //抛出异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
    try
    {
        if(!m_pRecordset->BOF)                               //判断指　是否在数据　最后
            m_pRecordset->MoveFirst();
```

```
        else
        {
            AfxMessageBox("表内数据为空");
            return false;
        }

        while(!m_pRecordset->adoEOF)
        {
            var = m_pRecordset->GetCollect("凭证号码");
            if(var.vt != VT_NULL)
                strregnumber = (LPCSTR)_bstr_t(var);
            m_addmoney_regnumberctr.AddString(strregnumber);
            m_pRecordset->MoveNext();                               //移动记录指　到下一条记录
        }
    }
    catch(_com_error *e)                                            //如果读数异常，给出提示
    {
        AfxMessageBox(e->ErrorMessage());
    }

    //关　记录
    m_pRecordset->Close();
    m_pRecordset = NULL;

    m_showuser=loguserid;
    //更新显示
    UpdateData(false);
    enable(0);
    return TRUE;
}
```

完成追加押金操作的“确定”按钮的处理函数，代码如下：

```
void CAddmoneydlg::OnOK()
{
    UpdateData(true);
//获得　入框内的　入数据
    m_pRecordsetout.CreateInstance(__uuidof(Recordset));

    CString strsqlstore;
    strsqlstore.Format("SELECT * FROM checkinregtable where 凭证号码='%s'",m_addmoney_regnumber);

    try                                                             //　接数据库
    {
        m_pRecordsetout->Open(_variant_t(strsqlstore),              //查询表中所有字段
        theApp.m_pConnection.GetInterfacePtr(),                     //获取数据库的 IDispatch 指
                            adOpenDynamic,
                            adLockOptimistic,
                            adCmdText);
```

```
    }
    catch(_com_error *e)                                           //捕获  接数据库异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
    try                                                            //更新数据库
    {
        float theaddedmoney=atof(m_addmoney_prehandmoney)+atof(m_addmoney);
            char strtheaddedmoney[50];
            _gcvt(theaddedmoney, 4, strtheaddedmoney);             //格式  换
            //写入数据表
        m_pRecordsetout->PutCollect("  收    ", _variant_t(strtheaddedmoney));

            m_pRecordsetout->Update();
            //更新数据库完毕
        AfxMessageBox("  加成功!");
    }
    catch(_com_error *e)                                           //捕获  接数据库异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
}
```

在追加押金时，需要登记一下，“登记”按钮处理的函数代码如下：

```
void CAddmoneydlg::OnCloseupCOMBOregnumber()
{
            _variant_t var;
    //使用 ADO 创建数据库记录
    m_pRecordset.CreateInstance(__uuidof(Recordset));

    //在 ADO 操作中建议语句中要常用 try...catch()来捕获  误信息

    UpdateData(true);

    m_addmoney_regnumberctr.GetWindowText(m_addmoney_regnumber);

    CString strsql;
    strsql.Format("SELECT * FROM checkinregtable where 凭证号码='%s'",m_addmoney_regnumber);
    try
    {//打开数据表
        m_pRecordset->Open(_variant_t(strsql),                     //查询表中所有字段
                        theApp.m_pConnection.GetInterfacePtr(),    //获取  接库的 IDispatch 指
                        adOpenDynamic,
                        adLockOptimistic,
                        adCmdText);
    }
    catch(_com_error *e)                                           //捕获异常
    {
```

```
        AfxMessageBox(e->ErrorMessage());
    }


    try
    {//判断指　是否在数据　最后
        if(!m_pRecordset->BOF)
            m_pRecordset->MoveFirst();
        else
        {
            AfxMessageBox("表内数据为空");
            return;
        }

        //从数据表中读取数据
            //读取姓名
            var = m_pRecordset->GetCollect("姓名");
            if(var.vt != VT_NULL)
                m_addmoney_name = (LPCSTR)_bstr_t(var);
            //读取房　号
            var = m_pRecordset->GetCollect("房　号");
            if(var.vt != VT_NULL)
                m_addmoney_roomnumber = (LPCSTR)_bstr_t(var);
            //读取客房类型
            var = m_pRecordset->GetCollect("客房类型");
            if(var.vt != VT_NULL)
                m_addmoney_roomlevel = (LPCSTR)_bstr_t(var);
            //读取客房价格
            var = m_pRecordset->GetCollect("客房价格");
            if(var.vt != VT_NULL)
                m_addmoney_roommoney = (LPCSTR)_bstr_t(var);
            //读取住宿天数
            var = m_pRecordset->GetCollect("住宿天数");
            if(var.vt != VT_NULL)
                m_addmoney_checkdays = atof((LPCSTR)_bstr_t(var));
            //读取住宿日期
            var = m_pRecordset->GetCollect("住宿日期");
            if(var.vt != VT_NULL)
                m_addmoney_indate = (LPCSTR)_bstr_t(var);
            //读取住宿时
            var = m_pRecordset->GetCollect("住宿时　");
            if(var.vt != VT_NULL)
                m_addmoney_intime = (LPCSTR)_bstr_t(var);
            //读取　收
            var = m_pRecordset->GetCollect("　收　　");
            if(var.vt != VT_NULL)
                m_addmoney_prehandmoney = (LPCSTR)_bstr_t(var);
           else
              m_addmoney_prehandmoney="000";
```

```
            //读取　宿日期
            var = m_pRecordset->GetCollect("　宿日期");
            if(var.vt != VT_NULL)
                m_addmoney_outdate = (LPCSTR)_bstr_t(var);
            //读取　宿时
            var = m_pRecordset->GetCollect("　宿时　");
            if(var.vt != VT_NULL)
                m_addmoney_outtime = (LPCSTR)_bstr_t(var);
            //读取提　日期
            var = m_pRecordset->GetCollect("提　日期");
            if(var.vt != VT_NULL)
                m_addmoney_alarmdate = (LPCSTR)_bstr_t(var);
            //读取提　时
            var = m_pRecordset->GetCollect("提　时　");
            if(var.vt != VT_NULL)
                m_addmoney_alarmtime  = (LPCSTR)_bstr_t(var);
            //更新显示
            UpdateData(false);

            //从数据库内读取数据完毕

    }
    catch(_com_error *e)
    {//如果读数异常，给出提示
        AfxMessageBox(e->ErrorMessage());
    }

        //关　记录
    m_pRecordset->Close();
    m_pRecordset = NULL;
    //更新显示
    UpdateData(false);
}
```

在追加押金时，需要将对应的输入框初始化设置，具体代码如下：

```
void CAddmoneydlg::Onaddmoney()
{
    //初始化　入框内容
    m_addmoney_regnumber = _T("");
    m_addmoney_name = _T("");
    m_addmoney_outdate = _T("");
    m_addmoney_outtime = _T("");
    m_addmoney_prehandmoney = _T("");
    m_addmoney_roomlevel = _T("");
    m_addmoney_roommoney = _T("");
    m_addmoney_roomnumber = _T("");
    m_addmoney_alarmdate = _T("");
    m_addmoney_alarmtime = _T("");
```



```
        m_addmoney_checkdays = 0.0f;
        m_addmoney_indate = _T("");
        m_addmoney_intime = _T("");
        m_addmoney = _T("");
        UpdateData(false);

}
```

其他函数的处理代码见源程序。

3.8　调房登记模块设计



3.8.1　调房登记模块概

调房登记模块是为实现客户调房而设计的，有的客户可能在住宿期间要求调换房间，该模块可以通过选择原房间号码和目标房间号码实现调房操作，其运行界面如图 3.5 所示。

3.8.2　调房登记模块技术分析

调房登记模块根据所选择的房间号，在住宿登记表中查询相关记录。如果查询到记录，则将记录显示在窗体上，然后输入目标房间号记录，将记录保存到住宿登记表中。根据房间号读取相关的住宿信息的主要代码如下：

```
if(!m_pRecordset->BOF)                              //判断指　是否在数据　最后
        m_pRecordset->MoveFirst();
else
{
        AfxMessageBox("表内数据为空");
        return;
}

//从数据表中读取客房价格字段
var = m_pRecordset->GetCollect("客房价格");
if(var.vt != VT_NULL)
        m_changeroom_roommoney = (LPCSTR)_bstr_t(var);
```

3.8.3　调房登记模块实现　程

本模块使用的数据表：Checkinregtable

（1）选择 Insert→Resource 命令，打开添加资源界面，选择 Dialog 选项，单击 New 按钮，插入新的对话框。

（2）利用类向导为此对话框资源设置属性。在 Name 文本框中输入对话框类名，如 CChangeroomdlg，

在 Base class 下拉列表框中选择一个基类，这里为 CDialog，单击 OK 按钮创建对话框。

（3）在工作区的资源视图中选择新创建的对话框，向对话框中添加静态文本、下拉列表框、编辑框和按钮等资源。

各个控件的 ID 和属性设置如表 3.6 所示。

表 3.6　各控件的 ID 和属性设置

控件 ID	对 应 变	控件 ID	对 应 变
IDC_COMBO_destroom	m_destroomctr	IDC_changeroom_idnumber	m_changeroom_idnumber
IDC_COMBO_sourceroom	m_sourceroomctr	IDC_changeroom_name	m_changeroom_name
IDC_COMBO_sourceroom	m_sourceroom	IDC_changeroom_roommoney	m_changeroom_roommoney
IDC_COMBO_destroom	m_destroom	IDC_changeroomdlg_regnumber	m_changeroom_regnumber
IDC_changeroom_beizhu	m_changeroom_beizhu	IDC_STATICshowuser	m_showuser
IDC_changeroom_idkind	m_changeroom_idkind		

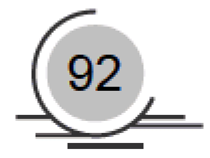
（4）在对应类的头文件 Changeroomdlg.h 中声明以下变量：

```
void enable(bool bEnabled);
CString destroomlevel;
_ConnectionPtr m_pConnection;
_CommandPtr m_pCommand;
_RecordsetPtr m_pRecordset;
_RecordsetPtr m_pRecordsetout;
```

该对话框类的初始化函数如下：

```
BOOL CChangeroomdlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //使用 ADO 创建数据库记录
    m_pRecordset.CreateInstance(__uuidof(Recordset));

        _variant_t var;
        CString strroomnumber;
    //在 ADO 操作中建议语句中要常用 try...catch()来捕获   误信息
    try
    {
    m_pRecordset->Open("SELECT * FROM checkinregtable",      //查询表中所有字段
    theApp.m_pConnection.GetInterfacePtr(),                  //获取   接库的 IDispatch 指
                        adOpenDynamic,
                        adLockOptimistic,
                        adCmdText);
    }
    catch(_com_error *e)                                     //捕获   接数据库异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
```



```
    try
    {
        if(!m_pRecordset->BOF)                                  //判断指　是否在数据　最后
            m_pRecordset->MoveFirst();
        else
        {
            AfxMessageBox("表内数据为空");
            return false;
        }

        //从数据库表中读取数据
        while(!m_pRecordset->adoEOF)
        {//读取房　号
            var = m_pRecordset->GetCollect("房　号");
            if(var.vt != VT_NULL)
            strroomnumber = (LPCSTR)_bstr_t(var);
            m_sourceroomctr.AddString(strroomnumber);          //添加到列表
            m_destroomctr.AddString(strroomnumber);
            m_pRecordset->MoveNext();                           //移动记录　指
        }
    }
    catch(_com_error *e)                                        //捕获异常
    {
        AfxMessageBox(e->ErrorMessage());
    }

    //关　记录
    m_pRecordset->Close();
    m_pRecordset = NULL;

    //获得操作员 ID
    m_showuser=loguserid;
    //显示更新
    UpdateData(false);
    enable(0);

    return TRUE;
}
```

完成调房登记模块的“确定”按钮的处理函数，代码如下：

```
void CChangeroomdlg::OnOK()
{
    UpdateData(true);
    m_pRecordsetout.CreateInstance(__uuidof(Recordset));
    CString strsqlstore;
    strsqlstore.Format("SELECT * FROM checkinregtable where 凭证号码='%s'",m_changeroom_regnumber);
    //打开数据库
   try
```

```
    {
        m_pRecordsetout->Open(_variant_t(strsqlstore),              //查询表中所有字段
          //获取   接库的 IDispatch 指
        theApp.m_pConnection.GetInterfacePtr(),
                                adOpenDynamic,
                                adLockOptimistic,
                                adCmdText);
    }
    catch(_com_error *e)
    {//捕获打开数据库时的异常情况，并给出提示
        AfxMessageBox(e->ErrorMessage());
    }
    try
    {
    //往数据库内写入数据
            CString zhaiyao;
    zhaiyao.Format("从原房   %s 调换到目标房   %s",m_sourceroom,m_destroom);
            m_pRecordsetout->PutCollect("房   号", _variant_t(m_destroom));
            //写入数据表“房   号”字段
            m_pRecordsetout->PutCollect("摘要", _variant_t(zhaiyao));
            //写入数据表“摘要”字段
m_pRecordsetout->PutCollect("客房价格", _variant_t(m_changeroom_roommoney));
            //写入数据表“客房价格”字段
            m_pRecordsetout->PutCollect("客房类型", _variant_t(destroomlevel));
            //写入数据表“客房类型”字段
            m_pRecordsetout->Update();
           //写入数据完毕，给出提示
           AfxMessageBox("调换成功!");
           //UpdateData(false);
    }
    catch(_com_error *e)//捕获写入数据时的异常情况，实时显示
    {
        AfxMessageBox(e->ErrorMessage());
    }
//CDialog::OnOK();
}
```

顾客要求调房，就需要提供证件等有效信息进行查询确认，实现的具体代码如下：

```
void CChangeroomdlg::OnCloseupCOMBOsourceroom()
{
     _variant_t var;
     //使用 ADO 创建数据库记录
     m_pRecordset.CreateInstance(__uuidof(Recordset));

     //在 ADO 操作中建议语句中要常用 try...catch()来捕获   误信息
     //获取   入框内容
     UpdateData(true);
```

```
CString strsql;
strsql.Format("SELECT * FROM checkinregtable where 房　号='%s'",m_sourceroom);
try
{//打开数据库
	m_pRecordset->Open(_variant_t(strsql),			//查询表中所有字段
		theApp.m_pConnection.GetInterfacePtr(),		//获取　接库的 IDispatch 指
		adOpenDynamic,
		adLockOptimistic,
		adCmdText);
}
catch(_com_error *e)
{//捕获异常
	AfxMessageBox(e->ErrorMessage());
}


try
{
	if(!m_pRecordset->BOF)						//判断指　是否在数据　最后
		m_pRecordset->MoveFirst();
	else
	{
		AfxMessageBox("表内数据为空");
		return;
	}

	//从数据表中读取姓名字段
	var = m_pRecordset->GetCollect("姓名");
	if(var.vt != VT_NULL)
		m_changeroom_name= (LPCSTR)_bstr_t(var);

	//从数据表中读取凭证号码字段
	var = m_pRecordset->GetCollect("凭证号码");
	if(var.vt != VT_NULL)
		m_changeroom_regnumber= (LPCSTR)_bstr_t(var);
	//从数据表中读取证件名称字段
	var = m_pRecordset->GetCollect("证件名称");
	if(var.vt != VT_NULL)
		m_changeroom_idkind   = (LPCSTR)_bstr_t(var);
	//从数据表中读取证件号码字段
	var = m_pRecordset->GetCollect("证件号码");
	if(var.vt != VT_NULL)
		m_changeroom_idnumber = (LPCSTR)_bstr_t(var);

	//从数据表中读取备注字段
	var = m_pRecordset->GetCollect("备注");
	if(var.vt != VT_NULL)
		m_changeroom_beizhu = (LPCSTR)_bstr_t(var);
```

```
        UpdateData(false);

        //更新显示

    }
    catch(_com_error *e)                                    //捕获异常
    {
        AfxMessageBox(e->ErrorMessage());
    }

    //关  记录
    m_pRecordset->Close();
    m_pRecordset = NULL;
    //更新显示
    UpdateData(false);
}
```

调房登记选择房间类型等相关信息，实现的具体代码如下：

```
void CChangeroomdlg::OnCloseupCOMBOdestroom()
{
    _variant_t var;
    //使用 ADO 创建数据库记录
    m_pRecordset.CreateInstance(__uuidof(Recordset));

    //在 ADO 操作中建议语句中要常用 try...catch()来捕获  误信息

    UpdateData(true);


    CString strsql;
    strsql.Format("SELECT * FROM checkinregtable where 房  号='%s'",m_destroom);
    try//打开数据库
    {
        m_pRecordset->Open(_variant_t(strsql),                  //查询表中所有字段
            theApp.m_pConnection.GetInterfacePtr(),             //获取  接库的 IDispatch 指
            adOpenDynamic,
            adLockOptimistic,
            adCmdText);
    }
    catch(_com_error *e)
    {//捕获打开数据库时可能发生的异常情况
        AfxMessageBox(e->ErrorMessage());
    }

    try
    {
```

```
            if(!m_pRecordset->BOF)                              //判断指　是否在数据　最后
                m_pRecordset->MoveFirst();
            else
            {
                AfxMessageBox("表内数据为空");
                return;
            }

            //从数据表中读取客房价格字段
            var = m_pRecordset->GetCollect("客房价格");
            if(var.vt != VT_NULL)
                m_changeroom_roommoney = (LPCSTR)_bstr_t(var);
            //从数据表中读取客房类型字段
            var = m_pRecordset->GetCollect("客房类型");
            if(var.vt != VT_NULL)
                destroomlevel = (LPCSTR)_bstr_t(var);

            //读取数据完毕，然后更新显示
            UpdateData(false);

            //更新显示完毕

        }
        catch(_com_error *e)                                    //捕获异常
        {
            AfxMessageBox(e->ErrorMessage());
        }

        //关　记录
        m_pRecordset->Close();
        m_pRecordset = NULL;

        UpdateData(false);                                      //更新显示
}
```

其他部分的代码见源程序。



3.9　客房销售报表模块设计

3.9.1　客房　售报表模块概

客房销售报表模块实现客房销售报表信息的管理，可以按照时间段对该时间段内客房销售统计报表的详细信息进行精确的查询。运行界面如图 3.18 所示。

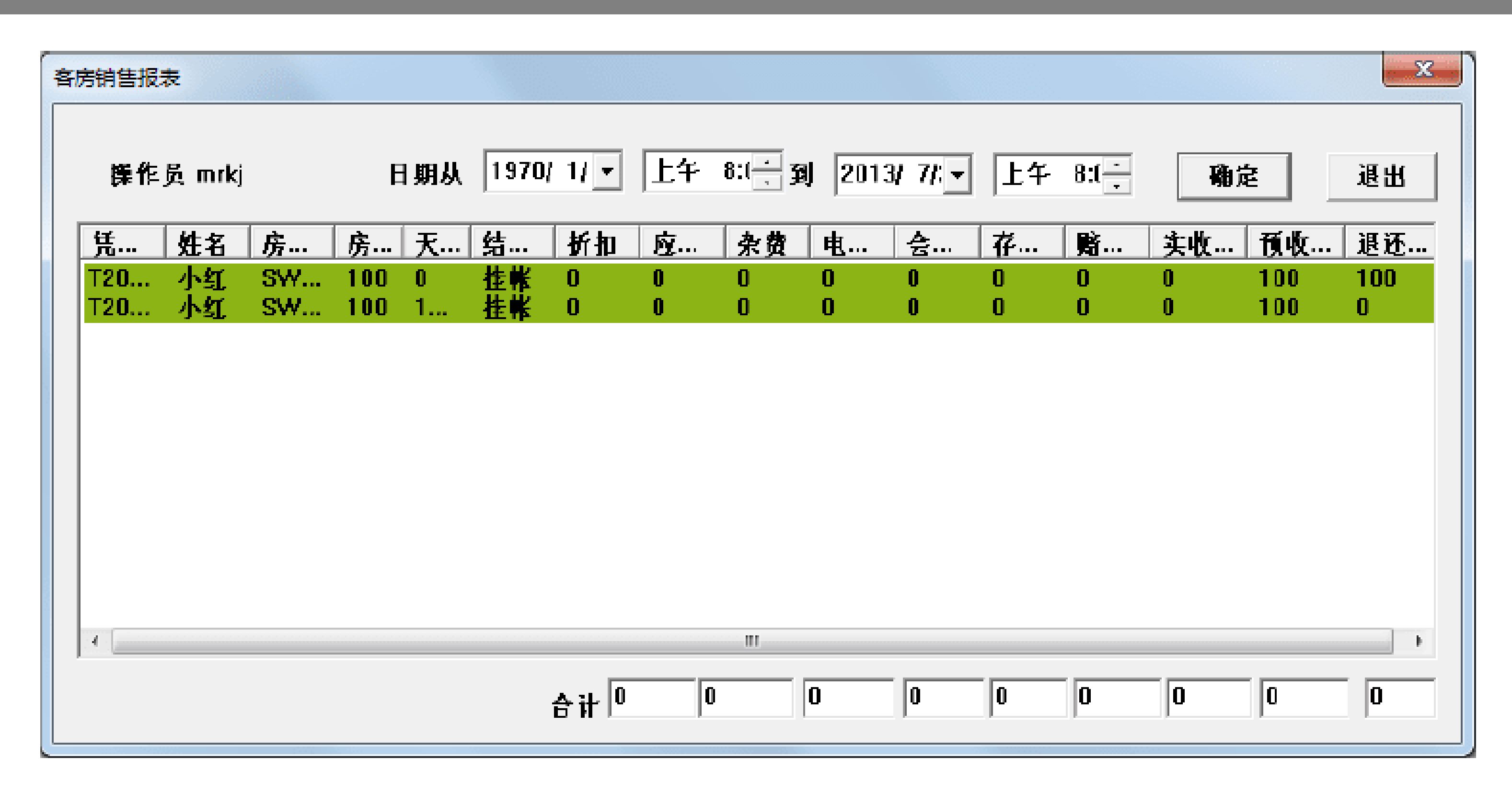


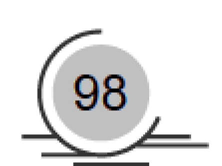
图 3.18 “客房销售报表”界面

3.9.2 客房 售报表模块技术分析

客房销售报表能根据指定的时间进行查询，查询的结果为指定时间内的数据信息。这是根据给定时间构造时间对象，与数据表中的时间进行比较，符合条件的就添加到窗体列表中，其实现关键代码如下：

```
        CString outyear,outmonth,outday;
        outyear=m_checkoutdate.Mid(0,4);
        outmonth=m_checkoutdate.Mid(5,m_checkoutdate.Find('-',6)-5);
        outday=m_checkoutdate.Mid(m_checkoutdate.ReverseFind('-')+1,
m_checkoutdate.GetLength()-m_checkoutdate.ReverseFind('-'));
         //构  时  对象
         CTime outtime(atoi(outyear),atoi(outmonth),atoi(outday),m_rooms
alebegintime.GetHour(),m_roomsalebegintime.GetMinute(),m_roomsalebegintime.GetSecond());
         //构  时  对象
         CTime begintime(m_roomsalebegindate.GetYear(),m_roomsale
begindate.GetMonth(),m_roomsalebegindate.GetDay(),m_roomsalebegintime.GetHour(),m_roomsalebegintime
.GetMinute(),m_roomsalebegintime.GetSecond());
         //构  时  对象
         CTime endtime(m_roomsaleenddate.GetYear(),m_roomsaleenddate.
GetMonth(),m_roomsaleenddate.GetDay(),m_roomsaleendtime.GetHour(),m_roomsaleendtime.GetMinute(),m
_roomsaleendtime.GetSecond());
```

3.9.3 客房 售报表模块实现 程


本模块使用的数据表：checkoutregtable

（1）选择 Insert→Resource 命令，打开添加资源界面，选择 Dialog 选项，单击 New 按钮，插入新的对话框。

（2）利用类向导为此对话框资源设置属性。在 Name 文本框中输入对话框类名，如 CRoomsaledlg，在 Base class 下拉列表框中选择一个基类，这里为 CDialog，单击 OK 按钮创建对话框。

（3）在工作区的资源视图中选择新创建的对话框，向对话框中添加静态文本、列表、编辑框、按钮和时间日期选择控件等资源。

界面中各个控件的 ID 和属性设置如表 3.7 所示。

表 3.7　窗体控件 ID 和属性设置

控件 ID	对应变　/标	控件 ID	对应变　/标
IDC_LIST_roomsale	m_roomsale_list	IDC_parkmoney	m_show_parkmoney
IDC_DATETIMEPICKERroomsale_endtime	m_roomsaleendtime	IDC_pregetroommoney	m_show_pregetroommoney
IDC_DATETIMEPICKERroomsale_enddate	m_roomsaleenddate	IDC_shouldgetmoney	m_show_shouldgetmoney
IDC_DATETIMEPICKERroomsale_begintime	m_roomsalebegintime	IDC_sumgetmoney	m_show_sumgetmoney
IDC_DATETIMEPICKERroomsale_begindate	m_roomsalebegindate	IDC_telmoney	m_show_telmoney
IDC_meetingmoney	m_show_meetingmoney	IDC_STATICshowuser	m_showuser
IDC_backroommoney	m_show_backroommoney	IDOK	确定
IDC_mendmoney	m_show_mendmoney	IDCANCEL	退出
IDC_mixmoney	m_show_mixmoney		

（4）在该对话框类对应的头文件 Roomsaledlg.h 中声明以下变量：

```
_ConnectionPtr m_pConnection;
_CommandPtr m_pCommand;
_RecordsetPtr m_pRecordset;
_RecordsetPtr m_pRecordsetfind;

CString m_realmoney;
CString m_room_money;
CString m_regnumber;
CString m_gustname;
CString m_gustaddr;
CString m_addr;
CString m_pre_discount;
CString m_roomlevel;
CString m_zhengjian_number;
CString m_checkinreg_reason;
CString m_discount_kind;
CString m_roomnumber;
CString m_zhengjian;
CString m_checkindate;
CString m_alarmdate;
CString m_alarmtime;
CString m_checkintime;
CString m_checkoutdate;
CString m_checkouttime;
CString m_checkdays;
CString m_discountnumber;
CString m_tel_money;
CString m_park_money;
```

```
CString m_mix_money;
CString m_mend_money;
CString m_meeting_money;
CString m_pre_handinmoney;
CString m_reback_money;
float sum_realmoney,sum_pregetmoney;
```

在该对话框类对应的源文件 Roomsaledlg.cpp 中，关键代码是“确定”按钮的处理函数。单击“确定”按钮可以统计选定时间段内的客房销售的详细信息，代码如下：

```
void CRoomsaledlg::OnOK()
{
        UpdateData(true);
    m_pRecordset.CreateInstance(__uuidof(Recordset));

        _variant_t var;
    m_roomsale_list.DeleteAllItems();
        int i=0;
    //在 ADO 操作中建议语句中要常用 try...catch()来捕获   误信息
    try
    {//打开数据表
m_pRecordset->Open("SELECT * FROM checkoutregtable",        //查询表中所有字段
    //获取   接库的 IDispatch 指
    theApp.m_pConnection.GetInterfacePtr(),
                        adOpenDynamic,
                        adLockOptimistic,
                        adCmdText);
    }
    catch(_com_error *e)                                      //抛出异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
    try
    {//判断指   是否在数据   最后
        if(!m_pRecordset->BOF)
            m_pRecordset->MoveFirst();
        else
        {
            AfxMessageBox("表内数据为空");
            return;
        }
        //从数据库表中读取数据
        while(!m_pRecordset->adoEOF)
        {
            //循环读取数据
            var = m_pRecordset->GetCollect("   房日期");
            if(var.vt != VT_NULL)
                m_checkoutdate = (LPCSTR)_bstr_t(var);
            CString outyear,outmonth,outday;
```

```
            outyear=m_checkoutdate.Mid(0,4);
            outmonth=m_checkoutdate.Mid(5,m_checkoutdate.Find('-',6)-5);
            outday=m_checkoutdate.Mid(m_checkoutdate.ReverseFind('-')+1,
m_checkoutdate.GetLength()-m_ checkoutdate.ReverseFind('-'));
            //构　时　对象
            CTime outtime(atoi(outyear),atoi(outmonth),atoi(outday),
m_roomsalebegintime.GetHour(),m_roomsalebegintime.GetMinute(),m_roomsalebegintime.GetSecond());
            //构　时　对象
            CTime begintime(m_roomsalebegindate.GetYear(),m_roomsalebegindate.
GetMonth(),m_roomsalebegindate.GetDay(),m_roomsalebegintime.GetHour(),m_roomsalebegintime.GetMinut
e(),m_roomsalebegintime.GetSecond());
            //构　时　对象
            CTime endtime(m_roomsaleenddate.GetYear(),m_roomsaleenddate.
GetMonth(),m_roomsaleenddate.GetDay(),m_roomsaleendtime.GetHour(),m_roomsaleendtime.GetMinute(),m
_roomsaleendtime.GetSecond());
            if((outtime<endtime)&&(outtime>begintime))
            {//满　条件的数据被读取，并在列表框内显示
            //读取数据表中“凭证号码”字段数据
            var = m_pRecordset->GetCollect("凭证号码");
            if(var.vt != VT_NULL)
                m_regnumber = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.InsertItem(i,m_regnumber.GetBuffer(50));
            //读取数据表中“姓名”字段数据
            var = m_pRecordset->GetCollect("姓名");
            if(var.vt != VT_NULL)
                m_gustname = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,1,m_gustname.GetBuffer(50));
            //读取数据表中“房　号”字段数据
            var = m_pRecordset->GetCollect("房　号");
            if(var.vt != VT_NULL)
                m_roomnumber = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,2,m_roomnumber.GetBuffer(50));
            //读取数据表中“客房价格”字段数据
            var = m_pRecordset->GetCollect("客房价格");
            if(var.vt != VT_NULL)
                m_room_money = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,3,m_room_money.GetBuffer(50));
            //读取数据表中“住宿天数”字段数据
            var = m_pRecordset->GetCollect("住宿天数");
            if(var.vt != VT_NULL)
                m_checkdays = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,4,m_checkdays.GetBuffer(50));
            //读取数据表中“折扣或招待”字段数据
            var = m_pRecordset->GetCollect("折扣或招待");
```

```
if(var.vt != VT_NULL)
    m_discount_kind = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,5,m_discount_kind.GetBuffer(50));
//读取数据表中“折扣”字段数据
var = m_pRecordset->GetCollect("折扣");
if(var.vt != VT_NULL)
    m_discountnumber = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,6,m_discountnumber.GetBuffer(50));
//读取数据表中“应收宿费”字段数据
var = m_pRecordset->GetCollect("应收宿费");
if(var.vt != VT_NULL)
    m_pre_discount = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,7,m_pre_discount.GetBuffer(50));
   m_show_shouldgetmoney=m_show_shouldgetmoney+atof(m_pre_discount);
//读取数据表中“杂费”字段数据
   var = m_pRecordset->GetCollect("杂费");
if(var.vt != VT_NULL)
    m_mix_money = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,8,m_mix_money.GetBuffer(50));
m_show_mixmoney=m_show_mixmoney+atof(m_mix_money);
//读取数据表中“电话费”字段数据
var = m_pRecordset->GetCollect("电话费");
if(var.vt != VT_NULL)
    m_tel_money = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,9,m_tel_money.GetBuffer(50));
m_show_telmoney=m_show_telmoney+atof(m_tel_money);
//读取数据表中“会议费”字段数据
var = m_pRecordset->GetCollect("会议费");
if(var.vt != VT_NULL)
    m_meeting_money = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,10,m_meeting_money.GetBuffer(50));
m_show_meetingmoney=m_show_meetingmoney+atof(m_meeting_money);
//读取数据表中“存  费”字段数据
var = m_pRecordset->GetCollect("存  费");
if(var.vt != VT_NULL)
    m_park_money = (LPCSTR)_bstr_t(var);
//在列表内显示该字段内容
m_roomsale_list.SetItemText(i,11,m_park_money.GetBuffer(50));
m_show_parkmoney=m_show_parkmoney+atof(m_park_money);
//读取数据表中“赔偿费”字段数据
var = m_pRecordset->GetCollect("赔偿费");
if(var.vt != VT_NULL)
    m_mend_money = (LPCSTR)_bstr_t(var);
```

```
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,12,m_mend_money.GetBuffer(50));
            m_show_mendmoney=m_show_mendmoney+atof(m_mend_money);
            //读取数据表中“    总计”字段数据
            var = m_pRecordset->GetCollect("    总计");
            if(var.vt != VT_NULL)
                m_realmoney = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,13,m_realmoney.GetBuffer(50));
                m_show_sumgetmoney=m_show_sumgetmoney+atof(m_realmoney);
            //读取数据表中“  收宿费”字段数据
            var = m_pRecordset->GetCollect("  收宿费");
            if(var.vt != VT_NULL)
                m_pre_handinmoney = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,14,m_pre_handinmoney.GetBuffer(50));
            m_show_pregetroommoney=m_show_pregetroommoney+atof(m_pre_handinmoney);
            //读取数据表中“    宿费”字段数据
            var = m_pRecordset->GetCollect("    宿费");
            if(var.vt != VT_NULL)
                m_reback_money = (LPCSTR)_bstr_t(var);
            //在列表内显示该字段内容
            m_roomsale_list.SetItemText(i,15,m_reback_money.GetBuffer(50));
            m_show_backroommoney=m_show_backroommoney+atof(m_reback_money);

            i++;                                     //移动记录  指  到下一条记录
            m_pRecordset->MoveNext();
            }
    else//如果不满  条件就直接    此记录
            {
                m_pRecordset->MoveNext();
                continue;
            }
        }
    }
    catch(_com_error *e)                             //捕获异常
    {
        AfxMessageBox(e->ErrorMessage());
    }
    //关  记录
    m_pRecordset->Close();
    m_pRecordset = NULL;
    UpdateData(false);
//CDialog::OnOK();
}
```

在 Roomsaledlg.cpp 源文件中，实现报表信息的代码如下：

```
BOOL CRoomsaledlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	CTime tTime;
	tTime=tTime.GetCurrentTime();
	//设置　认时
	m_roomsaleenddate=tTime;
	//TODO: Add extra initialization here
	//设置列表框　色
	m_roomsale_list.SetTextColor(RGB (0, 0, 0));
	m_roomsale_list.SetTextBkColor(RGB (140, 180, 20));
	//初始化列表框
	m_roomsale_list.InsertColumn(1,"凭证号码");
	m_roomsale_list.InsertColumn(2,"姓名");
	m_roomsale_list.InsertColumn(3,"房　号");
	m_roomsale_list.InsertColumn(4,"房价");
	m_roomsale_list.InsertColumn(5,"天数");
	m_roomsale_list.InsertColumn(6,"结款方式");
	m_roomsale_list.InsertColumn(7,"折扣");
	m_roomsale_list.InsertColumn(8,"应收宿费");
	m_roomsale_list.InsertColumn(9,"杂费");
	m_roomsale_list.InsertColumn(10,"电话费");
	m_roomsale_list.InsertColumn(11,"会议费");
	m_roomsale_list.InsertColumn(12,"存　费");
	m_roomsale_list.InsertColumn(13,"赔偿费");
	m_roomsale_list.InsertColumn(14,"实收　　");
	m_roomsale_list.InsertColumn(15,"　收宿费");
	m_roomsale_list.InsertColumn(16,"　　宿费");
	RECT rect;
	m_roomsale_list.GetWindowRect(&rect);
	int wid=rect.right-rect.left;
	int i=0;
	m_roomsale_list.SetColumnWidth(0,wid/16);
	m_roomsale_list.SetColumnWidth(1,wid/16);
	m_roomsale_list.SetColumnWidth(2,wid/16);
	m_roomsale_list.SetColumnWidth(3,wid/20);
	m_roomsale_list.SetColumnWidth(4,wid/20);
	m_roomsale_list.SetColumnWidth(5,wid/16);
	m_roomsale_list.SetColumnWidth(6,wid/16);
	m_roomsale_list.SetColumnWidth(7,wid/16);
	m_roomsale_list.SetColumnWidth(8,wid/16);
	m_roomsale_list.SetColumnWidth(9,wid/16);
	m_roomsale_list.SetColumnWidth(10,wid/16);
	m_roomsale_list.SetColumnWidth(11,wid/16);
	m_roomsale_list.SetColumnWidth(12,wid/16);
	m_roomsale_list.SetColumnWidth(13,wid/14);
	m_roomsale_list.SetColumnWidth(14,wid/14);
	m_roomsale_list.SetColumnWidth(15,wid/14);
	//设置列表框　格
```



```
 m_roomsale_list.SetExtendedStyle(LVS_EX_FULLROWSELECT);
                    //使用 ADO 创建数据库记录
 m_pRecordset.CreateInstance(__uuidof(Recordset));

      _variant_t var;

//在 ADO 操作中建议语句中要常用 try...catch 来捕获　误信息
try
{//打开数据表
      m_pRecordset->Open("SELECT * FROM checkoutregtable",          //查询表中所有字段
                                       theApp.m_pConnection.GetInterfacePtr(), //获取　接库的 IDispatch 指
                                       adOpenDynamic,
                                       adLockOptimistic,
                                       adCmdText);
}
catch(_com_error *e)
{//抛出异常情况
      AfxMessageBox(e->ErrorMessage());
}
try
{
      if(!m_pRecordset->BOF)//判断指　是否在数据　最后
            m_pRecordset->MoveFirst();
      else
      {
            AfxMessageBox("表内数据为空");
            return false;
      }

      while(!m_pRecordset->adoEOF)
      {//循环读取数据
            //读取数据表中“凭证号码”字段数据
            var = m_pRecordset->GetCollect("凭证号码");
            if(var.vt != VT_NULL)
                  m_regnumber = (LPCSTR)_bstr_t(var);
            //在列表框内显示该字段内容
            m_roomsale_list.InsertItem(i,m_regnumber.GetBuffer(50));

            //读取数据表中“姓名”字段数据
            var = m_pRecordset->GetCollect("姓名");
            if(var.vt != VT_NULL)
                  m_gustname = (LPCSTR)_bstr_t(var);
            //在列表框内显示该字段内容
            m_roomsale_list.SetItemText(i,1,m_gustname.GetBuffer(50));

            //读取数据表中“房　号”字段数据
            var = m_pRecordset->GetCollect("房　号");
            if(var.vt != VT_NULL)
                  m_roomnumber = (LPCSTR)_bstr_t(var);
```

```
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,2,m_roomnumber.GetBuffer(50));

//读取数据表中“客房价格”字段数据
var = m_pRecordset->GetCollect("客房价格");
if(var.vt != VT_NULL)
	m_room_money = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,3,m_room_money.GetBuffer(50));

//读取数据表中“住宿天数”字段数据
var = m_pRecordset->GetCollect("住宿天数");
if(var.vt != VT_NULL)
	m_checkdays = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,4,m_checkdays.GetBuffer(50));
//读取数据表中“折扣或招待”字段数据
var = m_pRecordset->GetCollect("折扣或招待");
if(var.vt != VT_NULL)
	m_discount_kind = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,5,m_discount_kind.GetBuffer(50));

//读取数据表中“折扣”字段数据
var = m_pRecordset->GetCollect("折扣");
if(var.vt != VT_NULL)
	m_discountnumber = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,6,m_discountnumber.GetBuffer(50));
//读取数据表中“应收宿费”字段数据
var = m_pRecordset->GetCollect("应收宿费");
if(var.vt != VT_NULL)
	m_pre_discount = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,7,m_pre_discount.GetBuffer(50));
//读取数据表中“杂费”字段数据
var = m_pRecordset->GetCollect("杂费");
if(var.vt != VT_NULL)
	m_mix_money = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,8,m_mix_money.GetBuffer(50));
//读取数据表中“电话费”字段数据
var = m_pRecordset->GetCollect("电话费");
if(var.vt != VT_NULL)
	m_tel_money = (LPCSTR)_bstr_t(var);
//在列表框内显示该字段内容
m_roomsale_list.SetItemText(i,9,m_tel_money.GetBuffer(50));
//读取数据表中“会议费”字段数据
var = m_pRecordset->GetCollect("会议费");
```

```
			if(var.vt != VT_NULL)
				m_meeting_money = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,10,m_meeting_money.GetBuffer(50));
			//读取数据表中“存　费”字段数据
			var = m_pRecordset->GetCollect("存　费");
			if(var.vt != VT_NULL)
				m_park_money = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,11,m_park_money.GetBuffer(50));
			//读取数据表中“赔偿费”字段数据
			var = m_pRecordset->GetCollect("赔偿费");
			if(var.vt != VT_NULL)
				m_mend_money = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,12,m_mend_money.GetBuffer(50));
			//读取数据表中“　　总计”字段数据
			var = m_pRecordset->GetCollect("　　总计");
			if(var.vt != VT_NULL)
				m_realmoney = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,13,m_realmoney.GetBuffer(50));

			//读取数据表中“　收宿费”字段数据
			var = m_pRecordset->GetCollect("　收宿费");
			if(var.vt != VT_NULL)
				m_pre_handinmoney = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,14,m_pre_handinmoney.GetBuffer(50));
			//读取数据表中“　　宿费”字段数据
			var = m_pRecordset->GetCollect("　　宿费");
			if(var.vt != VT_NULL)
				m_reback_money = (LPCSTR)_bstr_t(var);
			//在列表框内显示该字段内容
			m_roomsale_list.SetItemText(i,15,m_reback_money.GetBuffer(50));


			i++;
			m_pRecordset->MoveNext();			//记录　指　下移一条记录
		}


	}
	catch(_com_error *e)					//抛出异常情况，提示用户
	{
		AfxMessageBox(e->ErrorMessage());
	}

	//关　记录
```

```
	m_pRecordset->Close();
	m_pRecordset = NULL;
	//变   赋值
		m_show_meetingmoney=0;
		m_show_backroommoney=0;
		m_show_mendmoney=0;
		m_show_mixmoney=0;
		m_show_parkmoney=0;
		m_show_pregetroommoney=0;
		m_show_shouldgetmoney=0;
		m_show_sumgetmoney=0;
		m_show_telmoney=0;

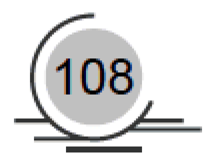
		m_showuser=loguserid;
	UpdateData(false);
	return TRUE;  //如果想要将焦点设置为控件就   回 TRUE
	              //另外，想要设置为 OCX 控件属性   就   回 FALSE
}
```

3.10　项目文件清单

客房管理系统项目文件清单如表 3.8 所示。

表 3.8　客房管理系统　目文件清单

文 件 名 称	文 件 类 型	文 件 描
Myhotel.dsp	工程文件	系统工程文件
Myhotel.dsw	工作区文件	系统工作区文件
Myhotel.rc	资源文件	系统资源文件
Myhotel Dlg.cpp	源文件	主对话框
Myhotel.cpp	源文件	系统应用程序
Addmoneydlg.cpp	源文件	追加押金窗体
ButtonST.cpp	源文件	按钮设置
Changerommdlg.cpp	源文件	调房登记窗体
Checkinregdlg.cpp	源文件	住宿登记窗体
CheckinregSET.cpp	源文件	住宿登记设置
Checkoutdlg.cpp	源文件	退宿结账
Findcheckindlg.cpp	源文件	住宿查询
Findcheckoutdlg.cpp	源文件	退宿查询
Findguazhangdlg.cpp	源文件	挂账查询
Findprebookroomdlg.cpp	源文件	客房预订查询
Findroomdlg.cpp	源文件	客房查询





续表

文 件 名 称	文 件 类 型	文 件 描
Findroomstatedlg.cpp	源文件	房态查看
Guesthandmoneydlg.cpp	源文件	客户结款
LoginDlg.cpp	源文件	用户登录
Reghandmoneydlg.cpp	源文件	登记预收报表
Repairpwdlg.cpp	源文件	密码设置
Resetdatabase.cpp	源文件	初始化设置
Roominfoset.cpp	源文件	入住信息设置
Roommoneyalarmdlg.cpp	源文件	宿费提醒
Roomprebookdlg.cpp	源文件	客房预订
Roomsaledlg.cpp	源文件	客房销售报表
Roomsalestaticdlg.cpp	源文件	客房销售统计
Setroomdlg.cpp	源文件	客房设置
Setuserabilitydlg.cpp	源文件	权限设置
Setusernamepwdlg.cpp	源文件	操作员代号和密码设置

3.11　本 章 总 结

本章的主要内容是根据酒店客房管理的实际情况设计一个管理系统。通过本章的学习，可以了解一个酒店客房管理系统的开发流程。本章通过详细的讲解及简洁的代码能使读者能够更快、更好地掌握数据库管理系统的开发技术，增加读者的实际开发能力和项目经验。

第 4 章

人事考勤管理系统

（Visual C++ 6.0+SQL Server 2014 实现）

对于一个小的企业，由于其员工不多，员工的出勤管理可能不是一个大问题。但随着企业不断壮大，员工不断增加，员工的出勤管理就会成为非常大的问题。人事考勤管理系统就是为了解决这个问题而产生的。有了人事考勤管理系统，就可以轻易地掌握每个员工的出勤状况。

通过学习本章，读者可以学到：

- 了解如何使用 ADO 连接数据库
- 掌握如何利用 ADO 封装类进行数据操作
- 了解数据库与程序间日期类型数据的操作
- 掌握如何使用 SQL 查询语句进行数据表的汇总查询



配置说明

4.1　开 发 背 景



××公司随着其业务的不断发展，公司的员工数量不断增加，人事考勤方面的管理已成为公司管理中的重要部分。传统的人事考勤制度已不能有效地管理员工的出勤状况，所以人事考勤系统就成为了人事考勤管理的有效工具。

4.2　需 求 分 析

在使用人事考勤管理系统时，用户需要输入用户名和密码进入人事考勤管理系统，从而对人事考勤管理系统的部门、员工的基本信息进行维护和管理。在考勤管理模块中录入员工当天的考勤信息，同时可对年、月和员工信息进行查询，还可通过考勤汇总查询功能对员工某月的考勤记录进行汇总，计算出员工月工作天数和早退、迟到的天数等。通过对人事考勤管理过程的研究和分析，要求本系统应该具有以下功能。

☑　用户登录。
☑　部门信息录入。
☑　人员信息管理。
☑　考勤信息录入。
☑　考勤信息汇总。

4.3　系 统 设 计

4.3.1　系统目标

人事考勤管理系统以实现员工日常出勤信息管理为设计目标，加以强大的数据库管理功能，便于对考勤信息进行管理，大大提高人事部门的日常工作效率。本系统在设计时应该满足以下几点。

☑　采用人机对话的操作方式，信息查询灵活、方便、快捷、准确，数据存储安全可靠。
☑　对考勤信息的操作简单，可以方便地进行添加、修改和删除。
☑　可以录入员工信息和部门信息。
☑　对员工的考勤信息可按月进行汇总计算。
☑　对用户输入的数据，系统进行严格的数据检验，尽可能排除人为的错误。
☑　系统最大限度地实现了易维护性和易操作性。
☑　系统运行稳定，安全可靠。

4.3.2　系统功能结构

人事考勤管理系统的功能结构图如图 4.1 所示。

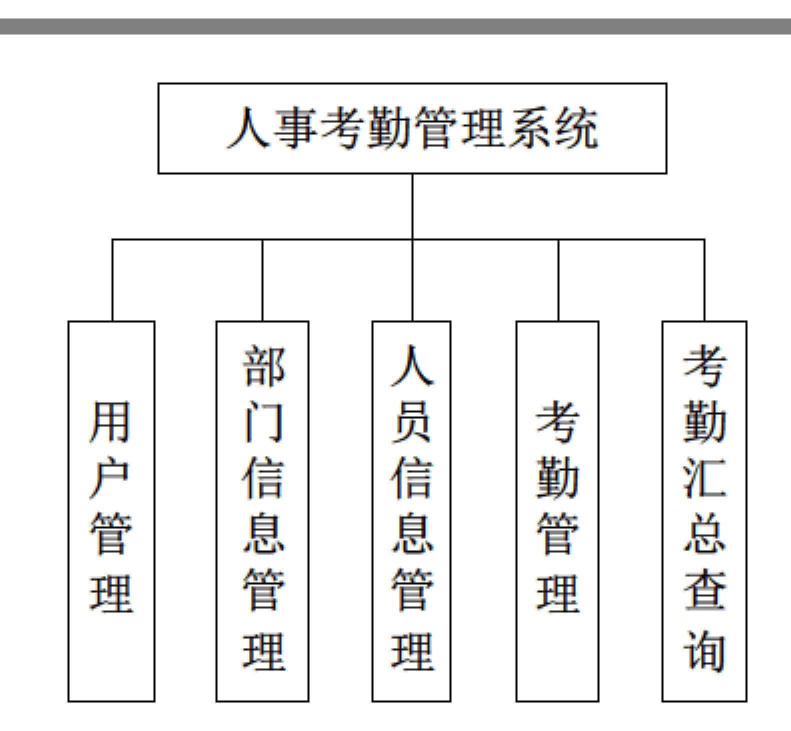


图 4.1　系统功能结构图

4.3.3　系统预览

人事考勤管理系统由多个功能模块组成，下面仅列出几个典型的功能模块，其他模块的实现请参见资源包中的源程序。

人事考勤管理系统的部门信息管理模块如图 4.2 所示，该模块用于管理各部门之间的结构信息。人员信息管理模块如图 4.3 所示，该模块用于维护员工的基本信息。



图 4.2　部门管理模块



图 4.3　人员信息管理模块

考勤管理模块如图 4.4 所示，该模块用于记录人事考勤的信息情况。考勤汇总查询模块如图 4.5 所示，该模块用于对员工的考勤信息进行汇总统计。



图 4.4　考勤管理模块

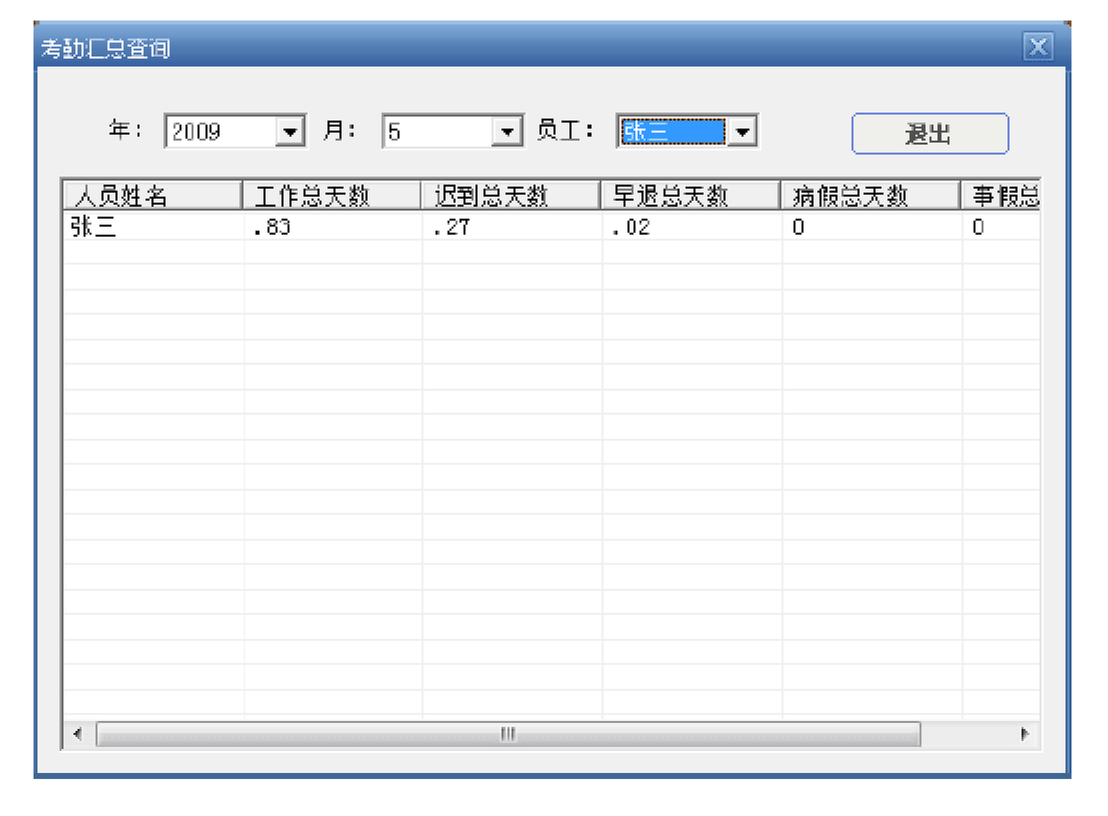
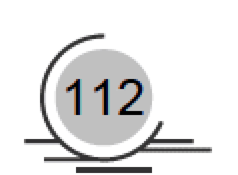


图 4.5　考勤汇总查询模块

4.3.4　业务流程图

人事考勤管理系统的业务流程图如图 4.6 所示。





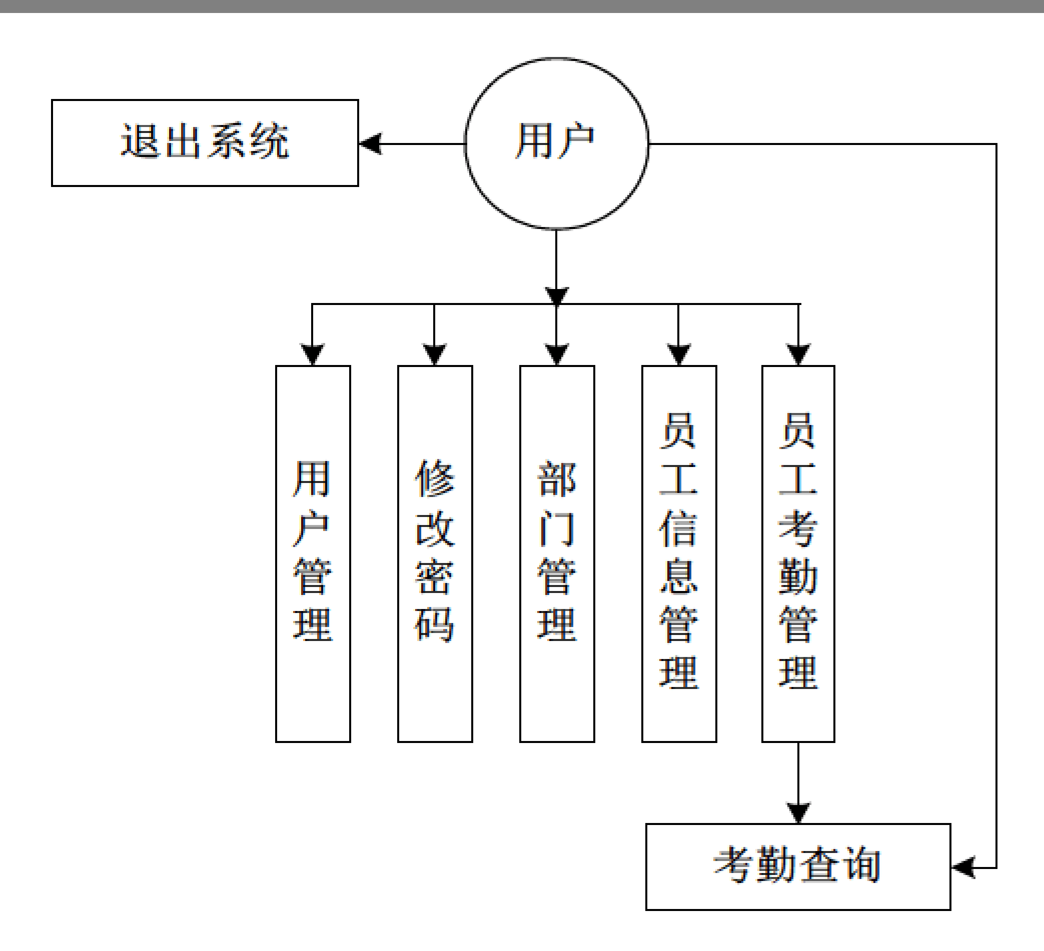


图 4.6　人事考勤管理系统的业务流程图

4.3.5　数据库设计

1．数据库分析

在人事考勤管理系统中使用了 SQL Server 2014 数据库来满足系统的需求，数据库名称为 tb_person，在数据库中创建 4 张表用于存储不同信息，如图 4.7 所示。

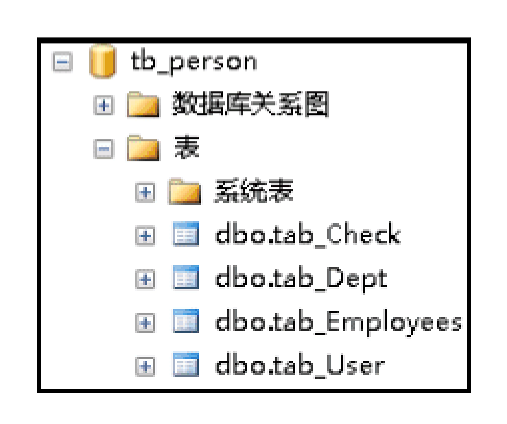


图 4.7　数据库中的表

2．数据库概念设计

根据前面介绍的需求分析和系统设计规划出本系统中使用的数据库实体对象，分别为管理员实体、部门实体、员工实体和考勤实体。下面将给出以上实体的 E-R 图。

（1）管理员实体

管理员实体包括编号、管理员姓名和密码信息。管理员实体 E-R 图如图 4.8 所示。

（2）部门实体

部门实体包括部门编号、部门名称、备注信息和上级部门编号。部门实体 E-R 图如图 4.9 所示。

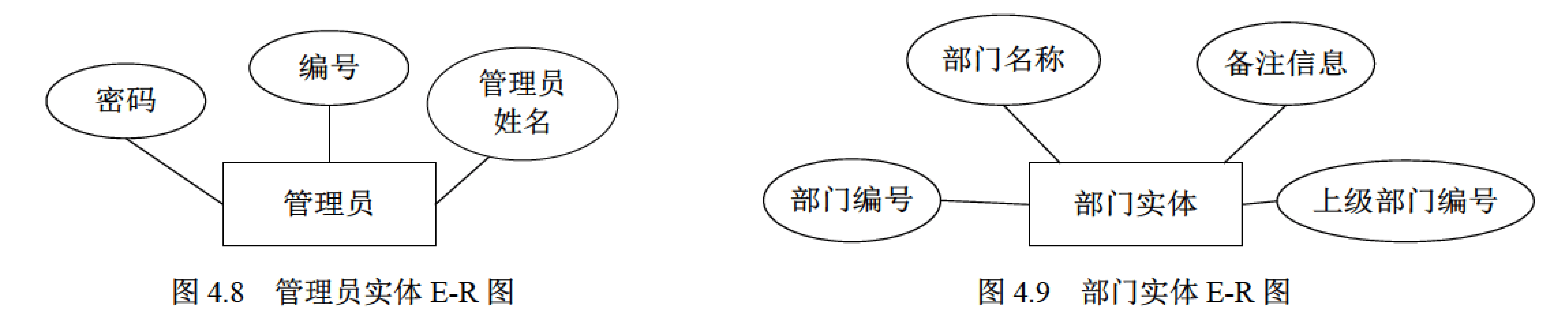
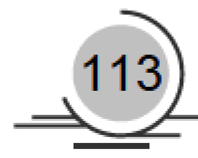


图 4.8　管理员实体 E-R 图　　图 4.9　部门实体 E-R 图

（3）员工实体

员工实体包括自动编号、员工编号、员工姓名、照片、性别和生日等信息。员工实体 E-R 图如图 4.10 所示。



（4）考勤实体

考勤实体包括人员姓名、考勤日期、上班时间、下班时间、上班考勤时间和下班考勤时间等信息。考勤实体 E-R 图如图 4.11 所示。

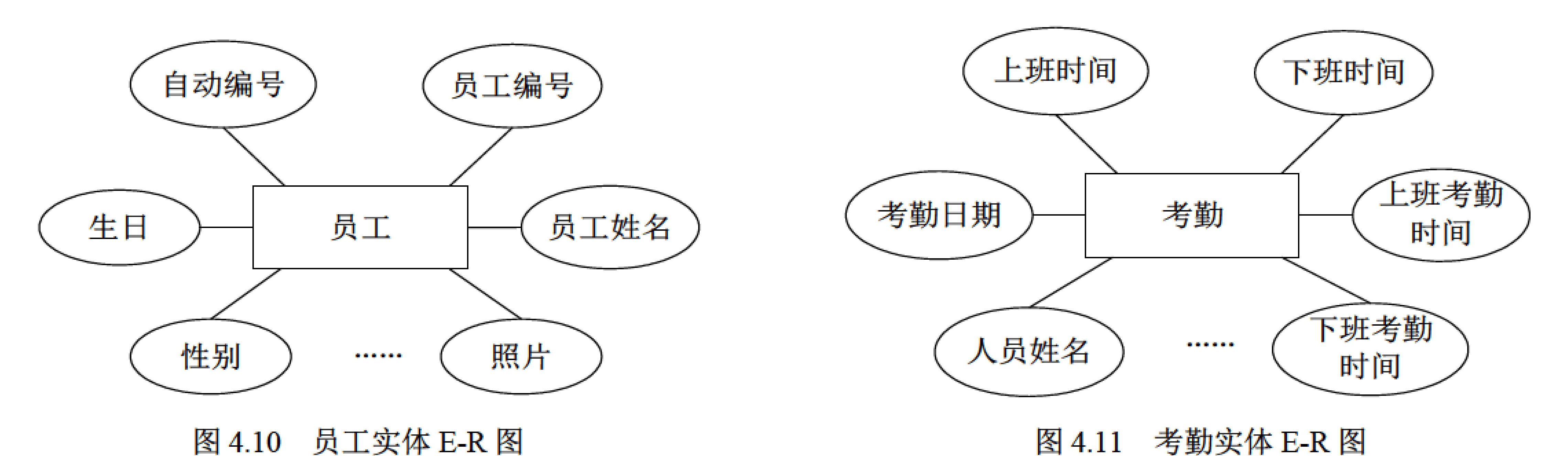


图 4.10　员工实体 E-R 图　　图 4.11　考勤实体 E-R 图

3．数据库逻辑结构设计

本例使用的是 SQL Server 2014 数据库，根据实体 E-R 图创建各数据表，下面给出人事考勤管理系统数据库中主要表的表结构。

（1）tab_User（管理员信息表）

管理员信息表用于保存管理员的信息，如图 4.12 所示。



MRWXK\MRWXK.tb_p...n - dbo.tab_User

列名	数据类型	允许 Null 值
ID	int	☐
UserName	varchar(50)	☐
PassWord	varchar(50)	☐

图 4.12　tab_User（管理员信息表）

（2）tab_Dept（部门信息表）

部门信息表用于记录部门的信息情况，如图 4.13 所示。

MRWXK\MRWXK.tb_p...n - dbo.tab_Dept

列名	数据类型	允许 Null 值
ID	int	☐
DeptName	varchar(50)	☐
Memo	varchar(50)	☑
PID	int	☐

图 4.13　tab_Dept（部门信息表）

（3）tab_Employees（员工信息表）

员工信息表用于保存公司员工的信息，如图 4.14 所示。

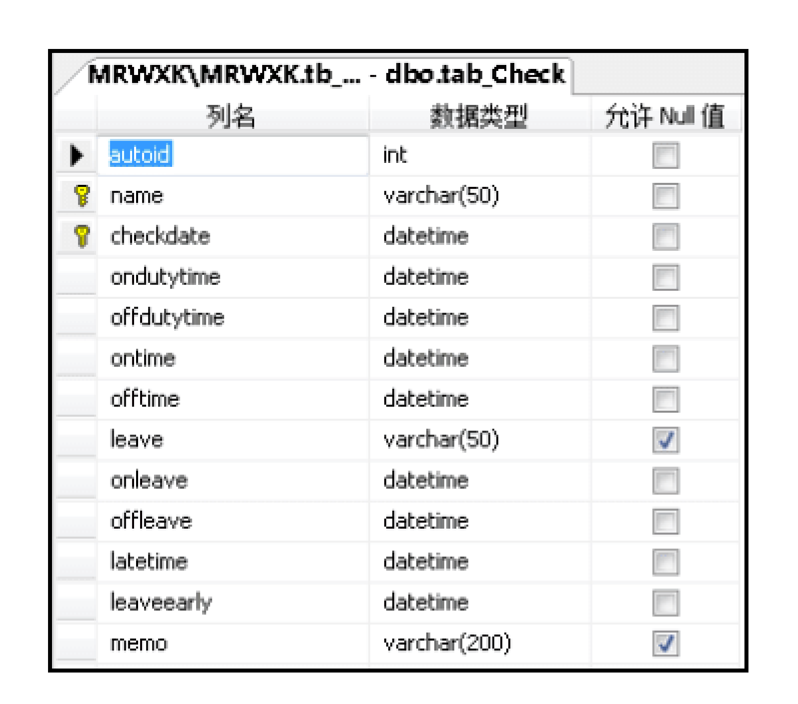
（4）tab_Check（考勤信息表）

考勤信息表用于记录员工每天的考勤信息，如图 4.15 所示。

MRWXK\MRWXK.tb_...bo.tab_Employees

列名	数据类型	允许 Null 值
AutoID	int	☐
Emp_Id	varchar(50)	☐
Emp_NAME	varchar(50)	☐
Photo	image	☑
Sex	char(2)	☑
Nationality	varchar(40)	☑
Birth	varchar(20)	☑
Political_Party	varchar(40)	☑
Culture_Level	varchar(40)	☑
Marital_Condition	varchar(20)	☑
Family_Place	varchar(60)	☑
Id_Card	varchar(20)	☑
Office_phone	varchar(30)	☑
Mobile	varchar(30)	☑
Files_Keep_Org	varchar(100)	☑
Hukou	varchar(100)	☑
HireDate	varchar(20)	☑
Dept	int	☑
Duty	varchar(40)	☑
Memo	varchar(200)	☑

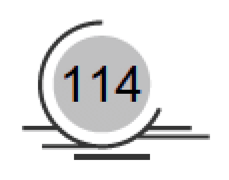
图 4.14　tab_Employees（员工信息表）



MRWXK\MRWXK.tb_... - dbo.tab_Check

列名	数据类型	允许 Null 值
autoid	int	☐
name	varchar(50)	☐
checkdate	datetime	☐
ondutytime	datetime	☐
offdutytime	datetime	☐
ontime	datetime	☐
offtime	datetime	☐
leave	varchar(50)	☑
onleave	datetime	☐
offleave	datetime	☐
latetime	datetime	☐
leaveearly	datetime	☐
memo	varchar(200)	☑

图 4.15　tab_Check（考勤信息表）





4.4　公共模块设计


视频讲解

本系统是使用 ADO 连接数据库的，为了能便于在程序中使用 ADO 建立数据库连接与数据表的操作，在公共类中对系统中所使用的 ADO 操作进行了封装。在该系统中建立了 ADO 的两个公共类 CADOConnection 和 CADODataSet，这两个类定义在 ADO.h 头文件中，实现在 ADO.cpp 文件中。

CADOConnection 类是用来连接数据库的，实现了对_Connection 接口的封装。CADOConneciton 类在头文件中的定义如下：

```
//载入 msado15dll，这样在工程中就不必再载入了
#import "msado15.dll" no_namespace rename("EOF","adoEOF")
class CADOConnection
{
private:
    static void InitADO();                                  //初始化 ADO
    static void UnInitADO();
protected:
    _ConnectionPtr m_Connection;                            //接口指针
public:
    BOOL IsOpen();                                          //判断是否与数据库连接
    _ConnectionPtr GetConnection();                         //获取连接接口
    CString GetSQLConStr(CString IP,CString DBName);         //获取 SQL 连接字符串
    BOOL Open(CString ConStr);                              //建立数据库连接
    CADOConnection();
    virtual ~CADOConnection();

};
CADOConnection * GetConnection();                           //获取全局连接类的函数
```

定义两个全局变量 ConCount 和 g_Connection，ConCount 变量是一个整型变量，用来记录在工程中所创建的 CADOConnection 类的实例个数。在构造方法中，当此变量为 0 时调用 CoInitialize 函数实现 OLE 的初始化。在析构方法中，当此变量为 0 时调用 CoUninitialize 方法取消 OLE 的初始化。

```
int ConCount = 0;
CADOConnection g_Connection;                                //全局数据库连接对象
```

GetConnection 函数是一个全局函数，用于返回全局数据库连接对象的指针，实现代码如下：

```
CADOConnection * GetConnection()
{
    return &g_Connection;
}
```

CADOConnection 方法是构造函数，用于初始化 OLE 和创建_Connection 接口的指针实例，实现代码如下：

```
CADOConnection::CADOConnection()
{
    InitADO();
    m_Connection.CreateInstance("ADODB.Connection");
}
```

~CADOConnection 方法是析构函数，用于取消 OLE 的初始化和放入_Connection 接口指针，实现代码如下：

```
CADOConnection::~CADOConnection()
{
    if (IsOpen())
        m_Connection->Close();
    m_Connection = NULL;
    UnInitADO();
}
```

InitADO 方法是一个静态方法，用于初始化 OLE，实现代码如下：

```
void CADOConnection::InitADO()
{
        if (ConCount++ == 0)
            CoInitialize(NULL);
};
```

UnInitADO 方法是一个静态方法，用于取消 OLE 的初始化，实现代码如下：

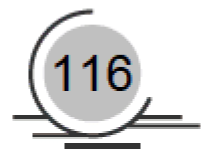
```
void CADOConnection::UnInitADO()
{
        if (--ConCount == 0)
            CoUninitialize();
};
```

Open 方法通过指定的数据库连接字符串与 SQL 数据库建立连接，实现代码如下：

```
BOOL CADOConnection::Open(CString ConStr)
{
    if (IsOpen())
        m_Connection->Close();
    m_Connection->Open((_bstr_t)ConStr,"","",adModeUnknown);
    return IsOpen();
}
```

GetSQLConStr 方法用来生成与数据库连接所需要的连接字符串，实现代码如下：

```
CString CADOConnection::GetSQLConStr(CString IP, CString DBName)
{
    CString Str;
    Str.Format("Provider=SQLOLEDB.1;Persist Security Info=False;\
```



```
			User ID=sa;Initial Catalog=%s;Data Source=%s",DBName,IP);
	return Str;
}
```

GetConnection 方法用于返回_Connection 接口指针，实现代码如下：

```
_ConnectionPtr CADOConnection::GetConnection()
{
	return m_Connection;
}
```

IsOpen 方法用来判断当前数据库连接对象与数据库的连接状态，实现代码如下：

```
BOOL CADOConnection::IsOpen()
{
	long State;
	m_Connection->get_State(&State);
	if (State == adStateOpen)
		return true;
	return false;
}
```

CADODataSet 类用来存储数据的数据集类，该类实现了_Recordset 接口的实例。该类在头文件中的定义如下：

```
class CADODataSet
{
protected:
	_RecordsetPtr m_DataSet;                                    //数据集接口指针
	CADOConnection *m_Connection;                               //数据库连接类对象
public:
	void Delete();                                              //记录删除
	int GetRecordNo();                                          //获取记录集行号
	void move(int nIndex);                                      //移动记录指针
	void Save();                                                //保存对记录集的修改
	void SetFieldValue(CString FieldName,_variant_t Value);     //设置字段的值
	void AddNew();                                              //添加新记录
	BOOL Next();                                                //记录集指针指向下一条记录
	FieldsPtr GetFields();                                      //获取记录集字段集合
	int GetRecordCount();                                       //获取记录集中记录数量
	void SetConnection(CADOConnection *pCon);                   //设置记录集的数据库连接对象
	BOOL Open(CString SQLStr);                                  //打开记录集

	CADODataSet();
	virtual ~CADODataSet();
private:
	BOOL IsOpen();                                              //判断记录集是否打开
};
```

CADODataSet 方法为记录集实现类的构造方法，在该方法中实现记录集接口对象的创建，实现代码如下：

```
CADODataSet::CADODataSet()
{
	m_DataSet.CreateInstance("ADODB.Recordset");
}
```

CADODataSet 类为记录集实现类的析构方法，在该方法中实现记录集的关闭与接口的释放，实现代码如下：

```
CADODataSet::~CADODataSet()
{
	if (IsOpen())
		m_DataSet->Close();
	m_DataSet = NULL;
	m_Connection = NULL;
}
```

SetConnection 方法用来设置记录集所连接的数据库连接类的对象，实现代码如下：

```
void CADODataSet::SetConnection(CADOConnection *pCon)
{
	m_Connection = pCon;
}
```

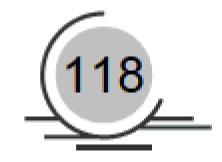
GetRecordCount 方法用来获取记录集中数据的数量，实现代码如下：

```
int CADODataSet::GetRecordCount()
{
	if (IsOpen())
		return m_DataSet->GetRecordCount();
	else
		return 0;
}
```

Open 方法通过 SQL 查询语句打开数据集，实现代码如下：

```
BOOL CADODataSet::Open(CString SQLStr)
{
	if (IsOpen())
		m_DataSet->Close();
	m_DataSet->Open(_bstr_t(SQLStr),
			_variant_t((IDispatch*)g_Connection.GetConnection(), true),
			adOpenKeyset, adLockUnspecified, adCmdText);
	return IsOpen();
}
```

IsOpen 方法用来判断数据集是否处于打开状态，实现代码如下：



```
BOOL CADODataSet::IsOpen()
{
	long State;
	m_DataSet->get_State(&State);
	if (State == adStateOpen)
		return true;

	return false;
}
```

GetFields 方法用来获取记录集中字段的集合，实现代码如下：

```
FieldsPtr CADODataSet::GetFields()
{
	return m_DataSet->GetFields();
}
```

Next 方法将记录集指针下移一位，实现代码如下：

```
BOOL CADODataSet::Next()
{
	if (m_DataSet->adoEOF)
		return false;
	m_DataSet->MoveNext();
	return true;
}
```

AddNew 方法用于向记录集中添加一个新行，实现代码如下：

```
void CADODataSet::AddNew()
{
	m_DataSet->AddNew();
}
```

SetFieldValue 方法用来向记录集中指定的字段赋值，实现代码如下：

```
void CADODataSet::SetFieldValue(CString FieldName, _variant_t Value)
{
	m_DataSet->PutCollect((_bstr_t)FieldName,Value);
}
```

Save 方法用来保存对记录集中数据所做的更改，实现代码如下：

```
void CADODataSet::Save()
{
	m_DataSet->Update();
}
```

Move 方法将记录集的当前指针移动到指定的索引位置，实现代码如下：

```
void CADODataSet::move(int nIndex)
{
	m_DataSet->MoveFirst();
	m_DataSet->Move(nIndex);
}
```

GetRecordNo 方法用来获取记录集中的当前行号，实现代码如下：

```
int CADODataSet::GetRecordNo()
{
	return m_DataSet->AbsolutePosition;
}
```

Delete 方法用来删除记录集中的当前行，实现代码如下：

```
void CADODataSet::Delete()
{
	m_DataSet->Delete(adAffectCurrent);
}
```



4.5　主窗体设计

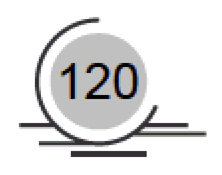
人事考勤管理系统主窗体由菜单和客户区域组成，其中，客户区域显示了一幅位图，主窗体效果如图 4.16 所示。



图 4.16　人事考勤管理系统的主窗体

主窗体设计步骤如下：

（1）启动 Visual C++ 6.0，选择 File→New 命令，打开 New 对话框。在 New 对话框左侧的列表视图中





选择 MFC AppWizard[exe]选项，在 Project name 文本框中输入工程名称，在 Location 文本框中设置工程保存的路径，如图 4.17 所示。

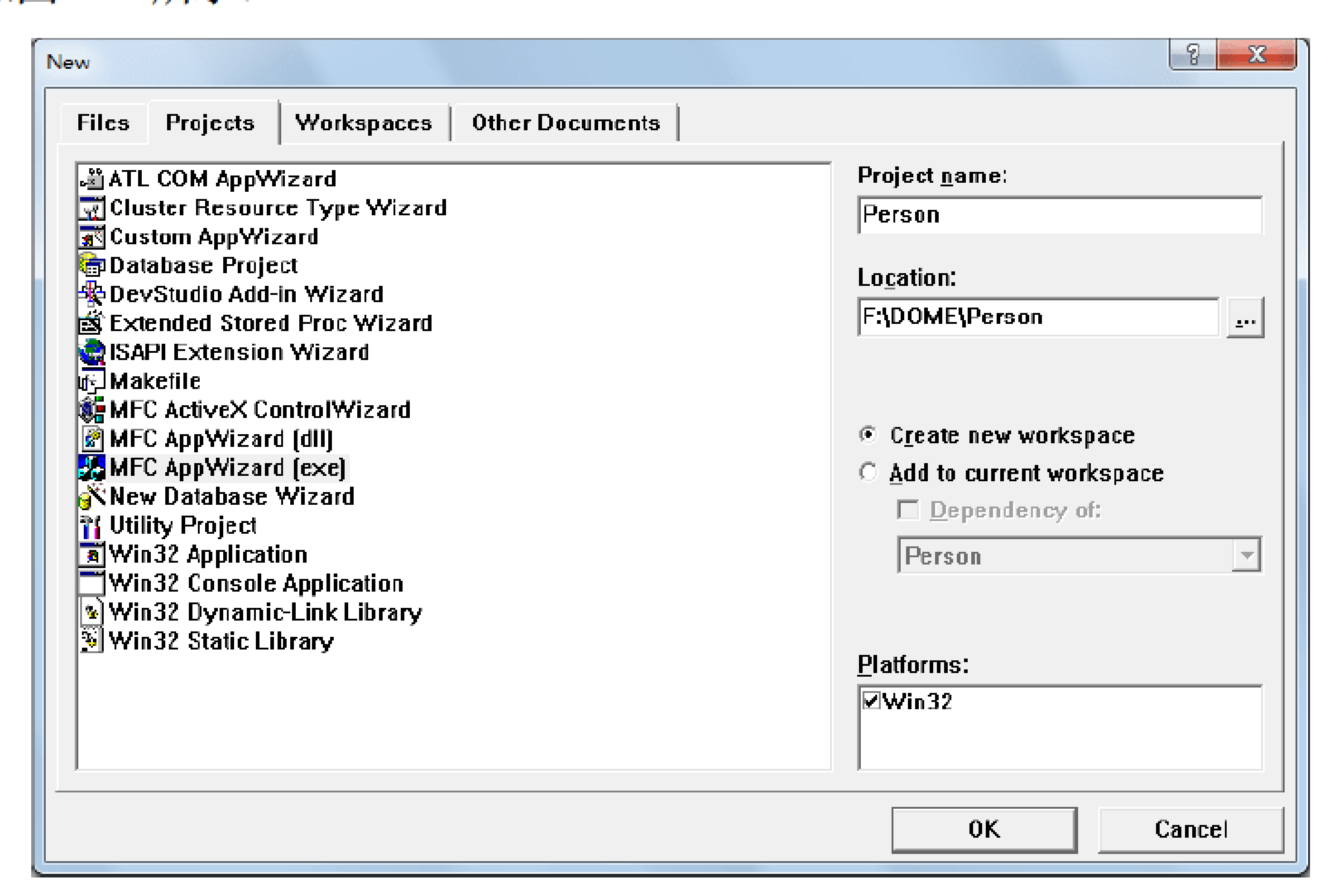


图 4.17　New 对话框

（2）单击 OK 按钮，进入 MFC AppWizard-Step1 对话框，选中 Dialog based 单选按钮，如图 4.18 所示。

（3）单击 Finish 按钮完成工程的创建。

（4）在工作区窗口的 ResourceView 视图中的节点处右击，在弹出的快捷菜单中选择 Insert 命令，弹出 Insert Resource 对话框，如图 4.19 所示。

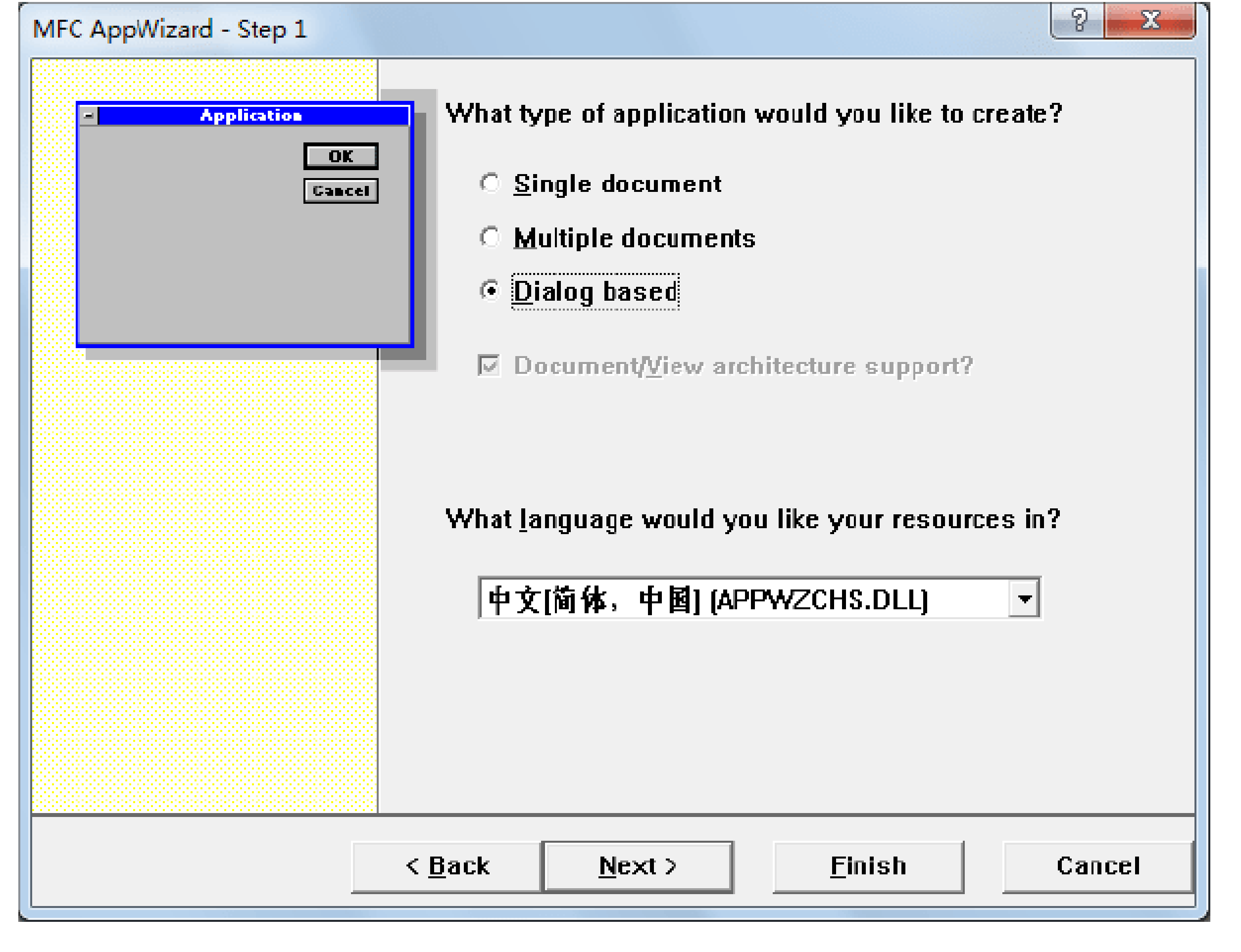


图 4.18　MFC AppWizard-Step1 对话框

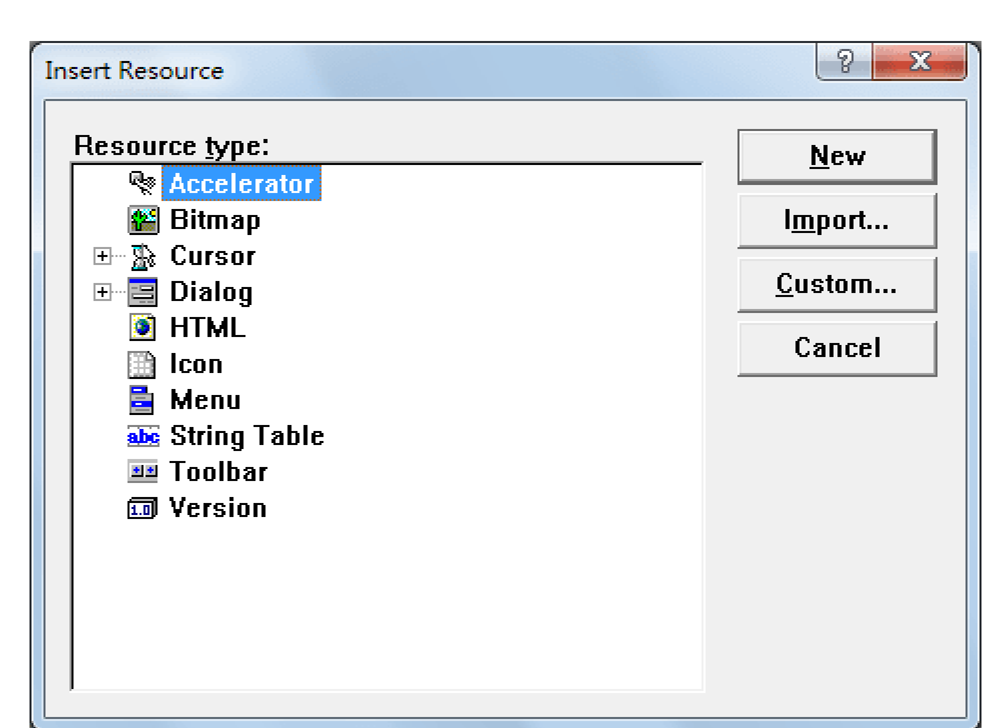


图 4.19　Insert Resource 对话框

（5）选择 Menu 选项，单击 New 按钮创建菜单资源，在 ResourceView 视图中双击新创建的菜单资源，编辑菜单资源，代码如下：

```
IDR_MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "系统设置"
    BEGIN
        MENUITEM "用户管理", ID_MENUUSER
        MENUITEM "修改密码", ID_MENUPASSWORD
        MENUITEM SEPARATOR
        MENUITEM "系统退出", ID_MENUEXIT
    END
    POPUP "基本信息管理"
    BEGIN
        MENUITEM "部门管理", ID_MENUDEPT
        MENUITEM "人员信息管理", ID_MENUPERSON
    END
    POPUP "员工考勤管理"
    BEGIN
        MENUITEM "考勤管理", ID_MENUCHECK
        MENUITEM "考勤汇总查询", ID_MENUCHECKSUM
    END
END
```



4.6　用户登录模块设计

4.6.1　用户登录模块概述

用户登录模块是所有管理系统所应具备的基础模块之一，该模块实现了用户登录系统时的检验功能，使没有权限的用户不能使用该系统，增加了系统的安全性。“登录”界面如图 4.20 所示。



图 4.20　“登录”界面

4.6.2　用户登录模块技术分析

用户登录窗体是整个系统中创建并显示的第一个窗体，所以该窗体应在主窗体创建前创建并显示。在用户登录窗体创建的同时应该创建数据库连接。这些操作都应在应用程序类的初始化方法中实现，该方法名为 InitInstance，代码如下：

```
BOOL CPersonApp::InitInstance()
{
    AfxEnableControlContainer();
#ifdef _AFXDLL
    Enable3dControls();
#else
    Enable3dControlsStatic();
```



```
#endif
	LoadSkin();
	//创建全局数据库连接
	BOOL bCon = GetConnection()->Open(GetConnection()->GetSQLConStr("127.0.0.1","tb_person"));
	CLoginDialog logindlg;                              //定义登录窗体对象
	if (logindlg.DoModal() != IDOK)                     //显示登录窗体
		return false;
	CPersonDlg dlg;                                     //定义应用程序主窗体
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();                      //显示主窗体
	if (nResponse == IDOK)
	{
	}
	else if (nResponse == IDCANCEL)
	{
	}
	return FALSE;
}
```

4.6.3　用户登录模块实现过程

本模块使用的数据表：tab_User

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGLOGIN，将对话框标题改为“登录”。

（2）向对话框中添加两个静态文本控件、一个编辑框控件、一个下拉列表框控件和两个按钮控件。分别设置两个静态文本控件的 Caption 属性为“用户名：”和“密码：”，设置编辑框控件的类型为 password，分别设置两个按钮控件的 Caption 属性为“确定”和“取消”。

（3）在窗体的初始化方法中创建用户表的数据集，并将用户名添加到下拉列表框控件中，代码如下：

```
BOOL CLoginDialog::OnInitDialog()
{
	CDialog::OnInitDialog();
	m_DataSet.SetConnection(GetConnection());           //设置数据集连接的数据库连接对象
	m_DataSet.Open("Select * From Tab_User");           //打开用户表
	int count = m_DataSet.GetRecordCount();             //获取用户数量
	for (int i = 0; i< count;i++)
	{                                                   //将用户名添加到下拉列表框控件中
		m_UserList.AddString((_bstr_t)m_DataSet.GetFields()->Item[L"UserName"]->Value);
		m_DataSet.Next();                               //记录下移
	}
	m_UserList.SetCurSel(0);                            //设置第一个用户为当前用户
	return TRUE;
}
```

（4）在“确定”按钮的事件中实现用户名和密码的验证，代码如下：

```
void CLoginDialog::OnLogin()
{
	CString sql,user,pass;
	m_UserList.GetWindowText(user);                         //获取用户名
	m_PassWord.GetWindowText(pass);                         //获取密码
	❶sql.Format("Select * From tab_User Where UserName = '%s' and PassWord = '%s'",
		user,pass);                                         //生成 SQL 查询语句
	m_DataSet.Open(sql);                                    //打开数据库
	if (m_DataSet.GetRecordCount() == 1)
	{
		::SetUserName(user);                                //设置当前用户
	❷	this->OnOK();
	}
	else
		AfxMessageBox("用户名或密码不正确！");
}
```

代码贴士

❶ Format 方法：用于格式化字符串。

❷ OnOK 方法：用于关闭当前窗口。


视频讲解

4.7 用户管理模块设计

4.7.1 用户管理模块概述

用户管理模块实现了对系统登录用户的添加、修改和删除操作，运行效果如图 4.21 所示。



图 4.21 用户管理模块的运行效果

4.7.2 用户管理模块技术分析

在用户管理模块中使用 CListCtrl 控件显示用户信息，当对某一记录进行编辑或删除操作时必须获取一个与记录对应的标识，所以在对用户列表进行添加时利用列表视图控件的 SetItemData 方法将记录集对应的行号添加到每一行对应的数据中。当对记录进行修改时就可以通过获取对应的行号对数据集中的数据进行修改。获取数据时使用列表视图控件中的 GetItemData 方法。

（1）SetItemData 方法用于设置与指定项相关的 32 位应用指定的值，语法如下：

```
BOOL SetItemData(int nItem,DWORD dwData)
```

☑ nItem：要设定数据的列表项的索引。

☑ dwData：与指定项相关联的 32 位值。



（2）GetItemData 方法用于获取与指定项相关的 32 位应用指定的值，语法如下：



```
DWORD GetItemData(int nItem) const
```

☑　nItem：要获取数据的列表项的索引值。

4.7.3　用户管理模块实现过程

本模块使用的数据表：tab_User

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGUSER，将对话框标题改为“用户管理”。

（2）向对话框中添加一个列表控件和 4 个按钮控件，各控件的属性设置如表 4.1 所示。

表 4.1　控件资源设置

控件 ID	控 件 属 性	对 应 变 量
IDC_LISTGRID	View：Report、Single selection	ClistCtrl m_grid
IDC_APPEND	Caption：添加	无
IDC_EDIT	Caption：修改	无
IDC_DELETE	Caption：删除	无
IDCANCEL	Caption：退出	无

（3）定义 UpdateGrid 方法，用来更新列表中显示的用户信息，实现代码如下：

```
void CUserManage::UpdateGrid()
{
     m_DataSet.Open("Select * From tab_User");                     //打开用户表
❶    m_grid.DeleteAllItems();                                       //清空列表中的全部记录
     for (int i = 0; i < m_DataSet.GetRecordCount();i++)           //循环记录集
     {
          //向列表视图中插入用户信息
❷         m_grid.InsertItem(i,(_bstr_t)m_DataSet.GetFields()->Item[L"UserName"]->Value);
          int no = m_DataSet.GetRecordNo();                         //获取当前记录集行号
          m_grid.SetItemData(i,no);                                 //存储列表中的项对应的行号
          m_DataSet.Next();                                         //行下移
     }
}
```

代码贴士

❶ DeleteAllItems 方法：用于删除当前列表控件中所有的列表项。

❷ InsertItem 方法：用于向列表控件中插入列表项。

（4）向对话框中添加 OnInitDialog 方法，在对话框的初始化方法中添加列表控件应显示的列头，并向列表控件中添加数据，代码如下：

```
BOOL CUserManage::OnInitDialog()
{
```

```
	CDialog::OnInitDialog();
	m_grid.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);	//列表控件样式
	m_grid.InsertColumn(0,"用户名");				//添加列
	m_grid.SetColumnWidth(0,150);					//设置列宽
	m_DataSet.SetConnection(::GetConnection());			//设置数据集的数据库连接对象
	UpdateGrid();							//向列表控件中添加数据
	return TRUE;
}
```

（5）在“添加”按钮的事件中弹出“用户编辑”窗体，输入用户名后单击“确定”按钮，实现对用户的添加，代码如下：

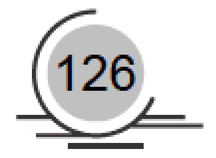
```
void CUserManage::OnAppend()
{
	CUserEdit useredit;						//定义“用户编辑”窗体
	if (useredit.DoModal() == IDOK)					//显示“用户编辑”窗体
	{
		m_DataSet.AddNew();					//数据集添加行
		m_DataSet.SetFieldValue("UserName",(_bstr_t)useredit.name);	//设置用户名字段的值为新用户
		m_DataSet.Save();					//保存数据集
		UpdateGrid();						//更新列表控件中的数据
	}
}
```

（6）在“修改”按钮的事件中弹出“用户编辑”窗体，输入用户名后单击“确定”按钮，实现对用户的修改，代码如下：

```
void CUserManage::OnEdit()
{
	CUserEdit useredit;						// “用户编辑”窗体
	int no = m_grid.GetItemData(m_grid.GetSelectionMark());		//获取当前行记录行号
	m_DataSet.move(no-1);						//记录集指向指定行
	useredit.name = (char *)(_bstr_t)m_DataSet.GetFields()->Item[L"UserName"]->Value;//获取用户名
	if (useredit.DoModal() == IDOK)					//显示“用户编辑”窗体
	{
		m_DataSet.SetFieldValue("UserName",(_bstr_t)useredit.name);	//设置新的用户名
		m_DataSet.Save();					//保存数据
		UpdateGrid();						//更新列表
	}
}
```

（7）在“删除”按钮的单击事件中获取当前记录进行删除操作，代码如下：

```
void CUserManage::OnDelete()
{
	if (MessageBox("是否删除此记录！ ","提示",
		MB_YESNO|MB_ICONWARNING) == IDYES)
	{
```



```
        int no = m_grid.GetItemData(m_grid.GetSelectionMark());    //获取记录集行号
        m_DataSet.move(no-1);                                      //移到指定行
        m_DataSet.Delete();                                        //删除
        m_DataSet.Save();                                          //保存
        UpdateGrid();                                              //更新列表
    }
}
```

4.7.4　单元测试

在测试用户管理模块时，曾出现这样的问题：用户在操作中不小心将所有的用户全部删除了，却没有创建新的用户，导致在下次登录时无法登录，下面来看一下原始的删除代码。

```
void CUserManage::OnDelete()
{
    if (MessageBox("是否删除此记录！","提示",
        MB_YESNO|MB_ICONWARNING) == IDYES)                        //弹出消息提示
    {
        //删除用户所选中的用户信息
        int no = m_grid.GetItemData(m_grid.GetSelectionMark());
        m_DataSet.move(no-1);
        m_DataSet.Delete();
        m_DataSet.Save();
        UpdateGrid();
    }
}
```

为了解决上述问题，可以设置一个超级用户，用户名为 mr，当要删除该用户时，提示不能删除，代码如下：

```
void CUserManage::OnDelete()
{
    int pos = m_grid.GetSelectionMark();                          //获得当前选中项索引
    if (pos != -1)
    {
        CString name = m_grid.GetItemText(pos,0);                 //获得当前选中项文本
        if (name != "mr")
        {
            if (MessageBox("是否删除此记录！","提示",
                MB_YESNO|MB_ICONWARNING) == IDYES)
            {
                int no = m_grid.GetItemData(m_grid.GetSelectionMark());
                m_DataSet.move(no-1);
                m_DataSet.Delete();
                m_DataSet.Save();
                UpdateGrid();
            }
```

```
			}
			else
			{
				MessageBox("该用户不能删除！");
				return;
			}
		}
}
```

视频讲解

4.8　部门管理模块设计

4.8.1　部门管理模块概述

部门管理模块记录了部门间的层次结构和部门信息，所以通常部门管理窗体中对于部门信息是使用树列表显示的。部门管理模块的运行效果如图 4.2 所示。

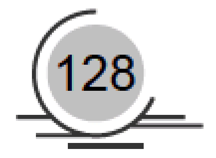
4.8.2　部门管理模块技术分析

由于部门通常都是存在层次级别的，所以在设计数据表结构时应至少创建 3 个字段：“编号”“父编号”“名称”。而在程序中显示部门信息时，也是根据“父编号”作为查询条件不断查找下一级部门的。

在本系统中，由于部门信息通常不会太多，所以可以用嵌套的方式将部门信息一次性读入树列表视图控件中，实现代码如下：

```
void CDeptManage::GetNode(HTREEITEM pNode,int nPid)
{
	HTREEITEM node;

	CADODataSet DataSet;                                          //定义记录集
	DataSet.SetConnection(::GetConnection());                     //设置数据库连接对象
	CString str;
	str.Format("Select * From tab_Dept where pid = %d",nPid);     //查询语句
	DataSet.Open(str);                                            //打开记录集
	int count = DataSet.GetRecordCount();                         //获取记录数量
	int ID;
	_variant_t value;
	for (int i = 0;i<count;i++)
	{
		node = m_tree.InsertItem((_bstr_t)DataSet.GetFields()->Item["DeptName"]->Value,pNode);   //部门名称
		value = (_variant_t)DataSet.GetFields()->Item["ID"]->Value;    //编号
		ID = value.intVal;
		m_tree.SetItemData(node,ID);                              //与节点关联
		GetNode(node,ID);                                         //获取子节点
```



```
            DataSet.Next();                                     //记录下移
        }
}
```

4.8.3　部门管理模块实现过程

本模块使用的数据表：tab_Dept

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGDEPT，将对话框标题改为“部门管理”。

（2）向对话框中添加一个树形视图控件和 4 个按钮控件，各控件的属性设置如表 4.2 所示。

表 4.2　控件资源设置

控件 ID	控 件 属 性	对 应 变 量
IDC_TREEDEPT	Has buttons、Has lines、Lines at root、Border	CTreeCtrl m_tree
IDC_APPEND	Caption：添加	无
IDC_EDIT	Caption：修改	无
IDC_DELETE	Caption：删除	无
IDCANCEL	Caption：关闭	无

（3）定义 GetNode 方法用来按层级关系获取部门表中的所有数据，并添加到树形视图控件中。该方法由 UpdateDept 方法进行调用，代码如下：

```
void CDeptManage::UpdateDept()
{
    m_tree.DeleteAllItems();                                    //清空树列表中的所有数据
    GetNode(TVI_ROOT,0);                                        //生成树列表
}

void CDeptManage::GetNode(HTREEITEM pNode,int nPid)
{
    HTREEITEM node;
    CADODataSet DataSet;                                        //定义记录集
    DataSet.SetConnection(::GetConnection());                   //设置数据库连接对象
    CString str;
    str.Format("Select * From tab_Dept where pid = %d",nPid);   //查询语句
    DataSet.Open(str);                                          //打开记录集
    int count = DataSet.GetRecordCount();                       //获取记录数量
    int ID;
    _variant_t value;
    for (int i = 0;i<count;i++)
    {
        node = m_tree.InsertItem((_bstr_t)DataSet.GetFields()->Item["DeptName"]->Value,pNode);//部门名称
        value = (_variant_t)DataSet.GetFields()->Item["ID"]->Value;     //编号
        ID = value.intVal;
```

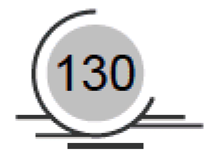
```
            m_tree.SetItemData(node,ID);                                //与节点关联
            GetNode(node,ID);                                           //获取子节点
            DataSet.Next();                                             //记录下移
        }
}
```

（4）当单击“添加”按钮时将弹出“部门编辑”窗体，输入部门信息后单击“确定”按钮将添加一个新的部门，代码如下：

```
void CDeptManage::OnAdd()
{
    CDeptEdit deptedit;                                                 // “部门编辑”窗体
    if (deptedit.DoModal() == IDOK)                                     //显示“部门编辑”窗体
    {
        HTREEITEM pNode = m_tree.GetSelectedItem();                     //获取选中节点
        int pID;
        if (deptedit.isroot)                                            //根节点
            pID = 0;
        else
            pID = m_tree.GetItemData(pNode);                            //子节点
        CADODataSet dataset;                                            //定义记录集
        dataset.SetConnection(::GetConnection());                       //设置数据库连接对象
        dataset.Open("Select top 1 * From tab_Dept");                   //打开记录集
        dataset.AddNew();                                               //添加新记录
        dataset.SetFieldValue("DeptName",(_variant_t)deptedit.name);    //部门名称
        dataset.SetFieldValue("memo",(_variant_t)deptedit.memo);        //备注
        dataset.SetFieldValue("PID",(long)pID);                         //父编号
        dataset.Save();                                                 //保存
        UpdateDept();                                                   //更新树列表
    }
}
```

（5）当单击“修改”按钮时将弹出“部门编辑”窗体，输入部门信息后单击“确定”按钮将添加一个新的部门，代码如下：

```
void CDeptManage::OnEdit()
{
    CDeptEdit deptedit;                                                 // “部门编辑”窗体
    deptedit.visible = false;
    HTREEITEM pNode = m_tree.GetSelectedItem();                         //获取选中节点
    if (pNode == 0)
        return;
    int pID = m_tree.GetItemData(pNode);                                //获取节点对应的编号
    CADODataSet dataset;                                                //定义记录集
    dataset.SetConnection(::GetConnection());                           //设置数据库连接
    CString str;
    str.Format("Select * From tab_Dept where id = %d",pID);             //生成查询语句
    dataset.Open(str);                                                  //打开记录集
```



```
        deptedit.name = (char *)(_bstr_t)dataset.GetFields()->Item[L"DeptName"]->Value;    //部门名称
        deptedit.memo = (char *)(_bstr_t)dataset.GetFields()->Item["memo"]->Value;    //备注
        if (deptedit.DoModal() == IDOK)                                               //显示“部门编辑”窗体
        {
                dataset.SetFieldValue("DeptName",(_variant_t)deptedit.name);          //部门名称
                dataset.SetFieldValue("memo",(_variant_t)deptedit.memo);              //备注
                dataset.Save();                                                       //保存
                UpdateDept();                                                         //更新树列表
        }
}
```

（6）当单击“删除”按钮时将删除当前选中的节点，代码如下：

```
void CDeptManage::OnDelete()
{
        HTREEITEM pNode = m_tree.GetSelectedItem();                                   //获取选中节点
        if (pNode == 0)
                return;
        if (MessageBox("是否删除此记录！","提示",
                MB_YESNO|MB_ICONWARNING) == IDYES)
        {
                int pID = m_tree.GetItemData(pNode);                                  //获取节点对应编号
                CADODataSet dataset;                                                  //定义记录集
                dataset.SetConnection(::GetConnection());                             //设置数据库连接
                CString str;
                str.Format("Select * From tab_Dept where id = %d",pID);               //生成查询语句
                dataset.Open(str);                                                    //打开记录集
                dataset.Delete();                                                     //删除记录
                dataset.Save();                                                       //保存
                UpdateDept();                                                         //更新树列表
        }
}
```

4.9　人员信息管理模块设计



4.9.1　人员信息管理模块概述

人员信息管理模块根据部门信息分类显示，同时可对人员信息进行维护，人员信息管理模块的运行效果如图 4.3 所示。

4.9.2　人员信息管理模块技术分析

在“人员信息管理”界面中可以看到，窗体的左侧是部门信息，右侧是人员信息。当选中某一部门信息分类时，右侧的人员信息会根据选中的部门进行人员信息的分类显示。这一操作主要是通过树

形视图控件中的 OnSelchanged 事件完成的，当树列表中的选中节点发生改变时就会触发该事件，代码如下：

```
void CPersonManage::OnSelchangedTreedept(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;		//获取树列表结构信息
	m_DeptID = m_tree.GetItemData(pNMTreeView->itemNew.hItem);	//获取部门编号
	UpdatePerson();							//更新人员信息
	*pResult = 0;
}
```

4.9.3 人员信息管理模块实现过程

本模块使用的数据表：tab_Dept、tab_Employees

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGPERSON，标题改为“人员信息管理”。

（2）向对话框中添加两个群组控件、一个树形视图控件、一个列表控件和 4 个按钮控件，各控件的属性设置如表 4.3 所示。

表 4.3 控件资源设置

控件 ID	控 件 属 性	对 应 变 量
IDC_LISTPERSON	View ：Icon、Single selection	ClistCtrl m_list
IDC_TREEDEPT	Has buttons、Has lines、Lines at root、Border	CTreeCtrl m_tree
IDC_APPEND	Caption：添加	无
IDC_EDIT	Caption：修改	无
IDC_DELETE	Caption：删除	无
IDCANCEL	Caption：退出	无

（3）添加 GetNode 方法获取部门表中的数据信息，并添加到树形视图控件中。该方法由 UpdateDept 方法调用，代码如下：

```
void CDeptManage::UpdateDept()
{
	m_tree.DeleteAllItems();					//清空树列表中的所有数据
	GetNode(TVI_ROOT,0);						//生成树列表
}

void CDeptManage::GetNode(HTREEITEM pNode,int nPid)
{
	HTREEITEM node;
	CADODataSet DataSet;						//定义记录集
	DataSet.SetConnection(::GetConnection());			//设置数据库连接对象
	CString str;
	str.Format("Select * From tab_Dept where pid = %d",nPid);	//查询语句
```



```
    DataSet.Open(str);                                          //打开记录集
    int count = DataSet.GetRecordCount();                       //获取记录数量
    int ID;
    _variant_t value;
    for (int i = 0;i<count;i++)
    {
        node = m_tree.InsertItem((_bstr_t)DataSet.GetFields()->Item["DeptName"]->Value,pNode);//部门名称
        value = (_variant_t)DataSet.GetFields()->Item["ID"]->Value;          //编号
        ID = value.intVal;
        m_tree.SetItemData(node,ID);                            //与节点关联
        GetNode(node,ID);                                       //获取子节点
        DataSet.Next();                                         //记录下移
    }
}
```

（4）定义 UpdatePerson 方法用来更新人员信息，将其显示在列表控件中，代码如下：

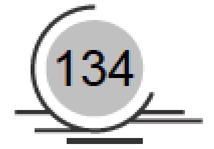
```
void CPersonManage::UpdatePerson()
{
    m_list.DeleteAllItems();                                    //清空列表中的数据
    CADODataSet DataSet;                                        //定义记录集
    DataSet.SetConnection(::GetConnection());                   //设置数据库连接对象
    CString str;
    if (m_DeptID == -1)
        str.Format("Select * From tab_Employees");              //显示所有人员信息
    else
        str.Format("Select * From tab_Employees where Dept = %d",m_DeptID);//显示指定部门的人员信息
    DataSet.Open(str);                                          //打开记录集
    int count = DataSet.GetRecordCount();                       //获取记录集记录数量
    int n = 0;
    _variant_t value;
    for (int i = 0;i<count;i++)                                 //循环记录集
    {
        int index = 1;
        m_list.InsertItem(n,(_bstr_t)DataSet.GetFields()->Item["Emp_Id"]->Value);//人员编号
        value = DataSet.GetFields()->Item["AutoID"]->Value;     //自动编号
        m_list.SetItemData(n,value.lVal);                       //将自动编号与列表中的项关联
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Emp_NAME"]->Value);//名称
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Sex"]->Value);     //性别
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Nationality"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Birth"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Political_Party"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Culture_Level"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Marital_Condition"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Id_Card"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Office_phone"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Mobile"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["HireDate"]->Value);
        m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Duty"]->Value);
```

```
            m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Memo"]->Value);
            m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Files_Keep_Org"]->Value);
            m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Hukou"]->Value);
            m_list.SetItemText(n,index++,(_bstr_t)DataSet.GetFields()->Item["Family_Place"]->Value);

            n++;
            DataSet.Next();                                         //记录下移
        }
}
```

（5）添加 OnInitDialog 方法，用于初始化“人员信息管理”对话框中的数据。在该方法中显示部门信息和人员信息，代码如下：

```
BOOL CPersonManage::OnInitDialog()
{
        CDialog::OnInitDialog();
        m_DeptID = -1;
        UpdateDept();
        int i = 0;
❶       m_list.InsertColumn(i,"人员编号");
❷       m_list.SetColumnWidth(i++,80);
        m_list.InsertColumn(i,"人员名称");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"性别");
        m_list.SetColumnWidth(i++,50);
        m_list.InsertColumn(i,"民族");
        m_list.SetColumnWidth(i++,50);
        m_list.InsertColumn(i,"出生日期");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"政治面貌");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"文化程度");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"婚姻状况");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"身份证号");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"办公电话");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"手机电话");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"到岗日期");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"职务");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"备注");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"家庭住址");
        m_list.SetColumnWidth(i++,100);
```



```
    m_list.InsertColumn(i,"档案所在地");
    m_list.SetColumnWidth(i++,100);
    m_list.InsertColumn(i,"户口所在地");
    m_list.SetColumnWidth(i++,100);
    m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
    UpdatePerson();
    return TRUE;
}
```

代码贴士

❶ InsertColumn 方法：用于向当前列表控件中插入列标题。

❷ SetColumnWidth 方法：用于设置列表控件的扩展风格。

（6）单击“添加”按钮，弹出“人员编辑”窗体，输入人员信息后单击“保存”按钮实现人员信息的添加，代码如下：

```
void CPersonManage::OnAdd()
{
    CPersonEdit personedit;
    personedit.m_DeptData = m_DeptID;
    if (personedit.DoModal() == IDOK)
    {
        CADODataSet dataset;
        dataset.SetConnection(::GetConnection());
        CString str = "select top 1 * from tab_Employees";
        dataset.Open(str);
        dataset.AddNew();
        dataset.SetFieldValue("Emp_Id",(_bstr_t)personedit.m_id);
        dataset.SetFieldValue("Emp_NAME",(_bstr_t)personedit.m_name);
        dataset.SetFieldValue("Sex",(_bstr_t)personedit.m_sex);
        dataset.SetFieldValue("Nationality",(_bstr_t)personedit.m_nationality);
        dataset.SetFieldValue("Birth",(_bstr_t)personedit.m_birth.Format("%Y-%m-%d"));
        dataset.SetFieldValue("Political_Party",(_bstr_t)personedit.m_farty);
        dataset.SetFieldValue("Culture_Level",(_bstr_t)personedit.m_culture);
        dataset.SetFieldValue("Marital_Condition",(_bstr_t)personedit.m_marital);
        dataset.SetFieldValue("Id_Card",(_bstr_t)personedit.m_card);
        dataset.SetFieldValue("Office_phone",(_bstr_t)personedit.m_office);
        dataset.SetFieldValue("Mobile",(_bstr_t)personedit.m_mobile);
        dataset.SetFieldValue("HireDate",(_bstr_t)personedit.m_hire.Format("%Y-%m-%d"));
        dataset.SetFieldValue("Duty",(_bstr_t)personedit.m_duty);
        dataset.SetFieldValue("Memo",(_bstr_t)personedit.m_memo);
        dataset.SetFieldValue("Files_Keep_Org",(_bstr_t)personedit.m_files);
        dataset.SetFieldValue("Hukou",(_bstr_t)personedit.m_hukou);
        dataset.SetFieldValue("Family_Place",(_bstr_t)personedit.m_family);
        dataset.SetFieldValue("dept",personedit.m_DeptData);
        dataset.Save();
        UpdatePerson();
    }
}
```

（7）单击“修改”按钮，弹出“人员编辑”窗体，输入人员信息后单击“保存”按钮实现人员信息的修改，代码如下：

```
void CPersonManage::OnEdit()
{
	if (m_list.GetSelectionMark() == -1)
		return;
	int id = m_list.GetItemData(m_list.GetSelectionMark());
	CPersonEdit personedit;
	CADODataSet dataset;
	dataset.SetConnection(::GetConnection());
	CString str;
	str.Format("select * from tab_Employees where autoid = %d",id);
	dataset.Open(str);
	personedit.m_id = (char *)(_bstr_t)dataset.GetFields()->Item["Emp_Id"]->Value;
	personedit.m_name = (char *)(_bstr_t)dataset.GetFields()->Item["Emp_NAME"]->Value;
	personedit.m_sex = (char *)(_bstr_t)dataset.GetFields()->Item["Sex"]->Value;
	personedit.m_nationality = (char *)(_bstr_t)dataset.GetFields()->Item["Nationality"]->Value;
	CString birth = (char *)(_bstr_t)dataset.GetFields()->Item["Birth"]->Value;
	if (!birth.IsEmpty())
	{
		//设置日期数据
		int yy=atoi(birth.Left(4));
		int mm=atoi(birth.Mid(6,2));
		int dd=atoi(birth.Mid(9,2));
		CTime tbirth(yy,mm,dd,0,0,0);
		personedit.m_birth = tbirth;
	}
	personedit.m_farty = (char *)(_bstr_t)dataset.GetFields()->Item["Political_Party"]->Value;
	personedit.m_culture = (char *)(_bstr_t)dataset.GetFields()->Item["Culture_Level"]->Value;
	personedit.m_marital = (char *)(_bstr_t)dataset.GetFields()->Item["Marital_Condition"]->Value;
	personedit.m_card = (char *)(_bstr_t)dataset.GetFields()->Item["Id_Card"]->Value;
	personedit.m_office = (char *)(_bstr_t)dataset.GetFields()->Item["Office_phone"]->Value;
	personedit.m_mobile = (char *)(_bstr_t)dataset.GetFields()->Item["Mobile"]->Value;
	CString hire = (char *)(_bstr_t)dataset.GetFields()->Item["HireDate"]->Value;
	if (!hire.IsEmpty())
	{
		//设置日期数据
		int yy=atoi(hire.Left(4));
		int mm=atoi(hire.Mid(6,2));
		int dd=atoi(hire.Mid(9,2));
		CTime thire(yy,mm,dd,0,0,0);
		personedit.m_hire = thire;
	}
	personedit.m_duty = (char *)(_bstr_t)dataset.GetFields()->Item["Duty"]->Value;
	personedit.m_memo = (char *)(_bstr_t)dataset.GetFields()->Item["Memo"]->Value;
	personedit.m_files = (char *)(_bstr_t)dataset.GetFields()->Item["Files_Keep_Org"]->Value;
	personedit.m_hukou = (char *)(_bstr_t)dataset.GetFields()->Item["Hukou"]->Value;
```

```
	personedit.m_family = (char *)(_bstr_t)dataset.GetFields()->Item["Family_Place"]->Value;
	personedit.m_DeptData = dataset.GetFields()->Item["Dept"]->Value;
	if (personedit.DoModal() == IDOK)
	{
		dataset.SetFieldValue("Emp_Id",(_bstr_t)personedit.m_id);
		dataset.SetFieldValue("Emp_NAME",(_bstr_t)personedit.m_name);
		dataset.SetFieldValue("Sex",(_bstr_t)personedit.m_sex);
		dataset.SetFieldValue("Nationality",(_bstr_t)personedit.m_nationality);
		dataset.SetFieldValue("Birth",(_bstr_t)personedit.m_birth.Format("%Y-%m-%d"));
		dataset.SetFieldValue("Political_Party",(_bstr_t)personedit.m_farty);
		dataset.SetFieldValue("Culture_Level",(_bstr_t)personedit.m_culture);
		dataset.SetFieldValue("Marital_Condition",(_bstr_t)personedit.m_marital);
		dataset.SetFieldValue("Id_Card",(_bstr_t)personedit.m_card);
		dataset.SetFieldValue("Office_phone",(_bstr_t)personedit.m_office);
		dataset.SetFieldValue("Mobile",(_bstr_t)personedit.m_mobile);
		dataset.SetFieldValue("HireDate",(_bstr_t)personedit.m_hire.Format("%Y-%m-%d"));
		dataset.SetFieldValue("Duty",(_bstr_t)personedit.m_duty);
		dataset.SetFieldValue("Memo",(_bstr_t)personedit.m_memo);
		dataset.SetFieldValue("Files_Keep_Org",(_bstr_t)personedit.m_files);
		dataset.SetFieldValue("Hukou",(_bstr_t)personedit.m_hukou);
		dataset.SetFieldValue("Family_Place",(_bstr_t)personedit.m_family);
		dataset.SetFieldValue("dept",personedit.m_DeptData);
		dataset.Save();
		UpdatePerson();
	}
}
```

（8）单击“删除”按钮，实现删除当前选中的人员信息记录的操作，代码如下：

```
void CPersonManage::OnDelete()
{
	if (MessageBox("是否删除此记录！","提示",
		MB_YESNO|MB_ICONWARNING) == IDYES)
	{
		if (m_list.GetSelectionMark() == -1)
			return;
		int id = m_list.GetItemData(m_list.GetSelectionMark());
		CADODataSet dataset;
		dataset.SetConnection(::GetConnection());
		CString str;
		str.Format("select * from tab_Employees where autoid = %d",id);
		dataset.Open(str);                                    //打开表
		dataset.Delete();                                     //删除记录
		dataset.Save();                                       //保存操作
		UpdatePerson();
	}
}
```


视频讲解

4.10 考勤管理模块设计

4.10.1 考勤管理模块概述

在考勤管理模块中可录入所有人员当天的考勤信息，并且可以根据年、月和人员信息对已录入的考勤记录进行查询。考勤管理模块的运行效果如图 4.4 所示。

4.10.2 考勤管理模块技术分析

在进行程序设计时，日期型数据可以使用字符串的形式存入日期类型的数据库字段中，但相反的，字符串类型的日期数据要想转换成日期类型的数据，就必须自己实现其转换功能。该模块实现了将字符串形式的日期和时间分别转换成日期类型的数据。

GetTimeForStr 方法用来将字符串形式的时间转换成日期类型，实现代码如下：

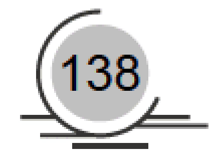
```
CTime CCheckManage::GetTimeForStr(CString timestr)
{
	int h,m,s;
	if (timestr.GetLength() < 8)                                    //不足 8 位补 0
		timestr = "0"+timestr;
	h = atoi(timestr.Left(2));                                      //获取小时
	m = atoi(timestr.Mid(3,2));                                     //获取分
	s = atoi(timestr.Right(2));                                     //获取秒
	CTime result(2000,1,1,h,m,s);                                   //生成日期
	return result;
}
```

GetDateForStr 方法用来将字符串类型的日期值转换成日期类型的数据，实现代码如下：

```
CTime CCheckManage::GetDateForStr(CString datestr)
{
	int y,m,d;
	y = atoi(datestr.Left(4));                                      //年
	m = atoi(datestr.Mid(5,2));                                     //月
	d = abs(atoi(datestr.Right(2)));                                //日
	CTime result(y,m,d,8,0,0);                                      //生成日期
	return result;
}
```

在该模块中还实现了一个时间相减的方法，在这个方法中实现的时间相减都是转换成秒后进行减法计算的，然后再将秒转换成对应的时间类型数据，实现代码如下：

```
CTime CCheckManage::DecTime(CTime one, CTime two)
{
```



```
    int yy,mm,dd,h,s,m,onetemp,twotemp;
    yy = 2000;//one.GetYear();//- two.GetYear();
    mm = 1;
    dd = 1;
    onetemp = one.GetSecond() + one.GetMinute() * 60 + one.GetHour() * 60 * 60;      //总秒数
    twotemp = two.GetSecond() + two.GetMinute() * 60 + two.GetHour() * 60 * 60;
    if ((onetemp - twotemp) < 0)
    {
        h = m = s = 0;
    }
    else
    {
        h = (onetemp - twotemp) / 60 / 60;                                           //小时
        m = ((onetemp - twotemp) - h * 60 * 60) / 60;                                //分钟
        s = ((onetemp - twotemp) - h * 60 * 60) - m * 60;                            //秒
    }
    CTime time (yy,mm,dd,h,m,s);                                                     //生成时间数据
    return time;
}
```

4.10.3　考勤管理模块实现过程

本模块使用的数据表：tab_Check

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGCHECK，标题改为“考勤管理”。

（2）向对话框中添加一个复选框控件、3 个静态文本框控件、3 个下拉列表框控件、4 个按钮控件和一个列表控件，各控件的属性设置如表 4.4 所示。

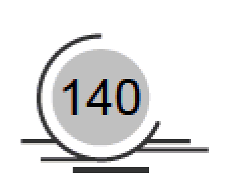
表 4.4　控件资源设置

控件 ID	控 件 属 性	对 应 变 量
IDC_CHECK1	Caption：全部显示	BOOL m_check
IDC_COMBOYY	Type：Drop List	CComboBox m_cyy CString m_yy
IDC_COMBOMM	Type：Drop List	CComboBox m_cmm CString m_mm
IDC_COMBOEMP	Type：Drop List	CComboBox m_cemp CString m_emp
IDC_LISTPERSON	View：Icon、Single selection	ClistCtrl m_list
IDC_APPEND	Caption：添加	无
IDC_EDIT	Caption：修改	无
IDC_DELETE	Caption：删除	无
IDCANCEL	Caption：退出	无

（3）添加 UpdateList 方法，用于显示人员考勤信息，实现代码如下：

```
void CCheckManage::UpdateList()
{
	this->UpdateData();
	CString str;
	if (m_check)
		str.Format("Select * From tab_Check");			//显示所有员工的考勤信息
	else
	{
		CString Starttime,EndTime;
		Starttime = m_yy + "-" + m_mm + "-1";
		EndTime.Format("DATEADD(month,1,'%s')",Starttime);
		if (m_emp == "(全部)")					//显示指定时间内的所有员工考勤信息
			str.Format("Select * From tab_Check where checkdate between '%s' and %s",Starttime,
EndTime);
		else							//显示指定时间内的某个员工考勤信息
			str.Format("Select * From tab_Check where name = '%s' and \
				checkdate between '%s' and %s",m_emp,Starttime,EndTime);
	}
	CADODataSet dataset;					//定义记录集
	dataset.SetConnection(::GetConnection());		//设置数据库连接对象
	dataset.Open(str);					//打开记录集
	m_list.DeleteAllItems();				//清空列表控件中的所有数据
	for (int i = 0; i < dataset.GetRecordCount(); i++)
	{
		int n = 0;
		long data = dataset.GetFields()->Item["autoid"]->Value;
		m_list.InsertItem(i,"");
		m_list.SetItemData(i,data);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["name"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["ondutytime"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["offdutytime"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["ontime"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["offtime"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["leave"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["onleave"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["offleave"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["latetime"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["leaveearly"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["memo"]->Value);
		m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["checkdate"]->Value);
		dataset.Next();					//记录下移
	}
}
```

（4）向对话框中添加 OnInitDialog 方法，在对话框初始化时设置列表控件的表头和列宽度以及查询条件选择控件，实现代码如下：



```
BOOL CCheckManage::OnInitDialog()
{
	CDialog::OnInitDialog();
	int i = 0;
	m_list.InsertColumn(i,"人员姓名");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"上班时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"下班时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"上班考勤时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"下班考勤时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"请假类别");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"请假起始时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"请假结束时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"迟到时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"早退时间");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"备注");
	m_list.SetColumnWidth(i++,100);
	m_list.InsertColumn(i,"考勤日期");
	m_list.SetColumnWidth(i++,130);
	m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
	m_check = true;
	this->UpdateData(false);
	int curyear,curmonth;                                   //当前年、月
❶	CTime time(CTime::GetCurrentTime());
❷	curyear = time.GetYear();                               //当前年
	curmonth = time.GetMonth();                             //当前月
	char value[10];
	for (int y = 2000; y < 2100;y++)                        //向“年”下拉列表框中添加数值
	{
		_itoa(y,value,10);
		m_cyy.InsertString(y-2000,value);
	}
	m_cyy.SetCurSel(curyear-2000);
	for (int n = 1; n<=12;n++)                              //向“月”下拉列表框中添加数值
	{
		_itoa(n,value,10);
		m_cmm.InsertString(n-1,value);
	}
	m_cmm.SetCurSel(curmonth-1);
	CADODataSet dataset;
```

```
    dataset.SetConnection(::GetConnection());
    dataset.Open("Select * From tab_Employees");                //打开员工信息表
    m_cemp.InsertString(0,"(全部)");
    for (int index = 1; index < dataset.GetRecordCount(); index++)    //向“员工”下拉列表框中添加员工信息
    {
        m_cemp.InsertString(index,(_bstr_t)dataset.GetFields()->Item["emp_name"]->Value);
        dataset.Next();
    }
    m_cemp.SetCurSel(0);
    UpdateList();                                                //更新考勤信息列表
    return TRUE;
}
```

代码贴士

❶ GetCurrentTime 方法：用于获取当前系统时间。

❷ GetYear 方法：用于获取 CTime 对象中的年数据。

（5）添加 OnAdd 方法，用于向考勤信息表中添加员工的日考勤数据，实现代码如下：

```
void CCheckManage::OnAdd()
{
    CCheckEdit checkedit;                                        //员工考勤编辑窗体
    if (checkedit.DoModal() == IDOK)                             //显示员工考勤编辑窗体
    {
        CString time;
        CString str = "Select top 1 * From tab_Check";
        CADODataSet dataset;
        dataset.SetConnection(::GetConnection());
        dataset.Open(str);                                       //打开员工考勤表
        dataset.AddNew();                                        //添加新记录
        dataset.SetFieldValue("name",(_bstr_t)checkedit.m_name);
        dataset.SetFieldValue("checkdate",(_bstr_t)checkedit.m_datecheck.Format("%Y-%m-%d"));
        dataset.SetFieldValue("ondutytime",(_bstr_t)checkedit.m_timeonduty.Format("%H:%M:%S"));
        dataset.SetFieldValue("offdutytime",(_bstr_t)checkedit.m_timeoffduty.Format("%H:%M:%S"));
        dataset.SetFieldValue("ontime",(_bstr_t)checkedit.m_timeon.Format("%H:%M:%S"));
        dataset.SetFieldValue("offtime",(_bstr_t)checkedit.m_timeoff.Format("%H:%M:%S"));
        dataset.SetFieldValue("leave",(_bstr_t)checkedit.m_leave);
        dataset.SetFieldValue("onleave",(_bstr_t)checkedit.m_timeonleave.Format("%H:%M:%S"));
        dataset.SetFieldValue("offleave",(_bstr_t)checkedit.m_timeoffleave.Format("%H:%M:%S"));
        dataset.SetFieldValue("memo",(_bstr_t)checkedit.m_memo);
        CTime latetime = DecTime(checkedit.m_timeon,checkedit.m_timeonduty);//时间相减
        time.Format("%d:%d:%d",latetime.GetHour(),latetime.GetMinute(),latetime.GetSecond());
        dataset.SetFieldValue("latetime",(_bstr_t)time);
        CTime leaveearly = DecTime(checkedit.m_timeoff,checkedit.m_timeoffduty);
        time.Format("%d:%d:%d",leaveearly.GetHour(),leaveearly.GetMinute(),leaveearly.GetSecond());
        dataset.SetFieldValue("leaveearly",(_bstr_t)time);
        dataset.Save();                                          //保存记录
        UpdateList();                                            //更新考勤信息列表
    }
}
```



（6）添加 OnEdit 方法，用于编辑考勤信息表中员工的日考勤数据，实现代码如下：

```
void CCheckManage::OnEdit()
{
	if (m_list.GetSelectionMark() == -1)							//判断是否存在选中记录
		return;
	int id = m_list.GetItemData(m_list.GetSelectionMark());			//获取记录唯一标识
	CCheckEdit checkedit;											//考勤信息编辑窗体
	CString str;
	str.Format("Select * From tab_Check where autoid = %d",id);
	CADODataSet dataset;
	dataset.SetConnection(::GetConnection());
	dataset.Open(str);												//打开记录集
	checkedit.m_name = (char *)(_bstr_t)dataset.GetFields()->Item["name"]->Value;
	checkedit.m_timeonduty = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["ondutytime"]->Value);
	checkedit.m_timeoffduty = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["offdutytime"]->Value);
	checkedit.m_timeon = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["ontime"]->Value);
	checkedit.m_timeoff = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["offtime"]->Value);
	checkedit.m_leave = (char *)(_bstr_t)dataset.GetFields()->Item["leave"]->Value;
	checkedit.m_timeonleave = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["onleave"]->Value);
	checkedit.m_timeoffleave = GetTimeForStr((char *)(_bstr_t)dataset.GetFields()->Item["offleave"]->Value);
	checkedit.m_memo = (char *)(_bstr_t)dataset.GetFields()->Item["memo"]->Value;
	checkedit.m_datecheck = GetDateForStr((char *)(_bstr_t)dataset.GetFields()->Item["checkdate"]->Value);
	if (checkedit.DoModal() == IDOK)								//显示考勤信息编辑窗体
	{
		CString time;
		dataset.SetFieldValue("name",(_bstr_t)checkedit.m_name);
		dataset.SetFieldValue("checkdate",(_bstr_t)checkedit.m_datecheck.Format("%Y-%m-%d"));
		dataset.SetFieldValue("ondutytime",(_bstr_t)checkedit.m_timeonduty.Format("%H:%M:%S"));
		dataset.SetFieldValue("offdutytime",(_bstr_t)checkedit.m_timeoffduty.Format("%H:%M:%S"));
		dataset.SetFieldValue("ontime",(_bstr_t)checkedit.m_timeon.Format("%H:%M:%S"));
		dataset.SetFieldValue("offtime",(_bstr_t)checkedit.m_timeoff.Format("%H:%M:%S"));
		dataset.SetFieldValue("leave",(_bstr_t)checkedit.m_leave);
		dataset.SetFieldValue("onleave",(_bstr_t)checkedit.m_timeonleave.Format("%H:%M:%S"));
		dataset.SetFieldValue("offleave",(_bstr_t)checkedit.m_timeoffleave.Format("%H:%M:%S"));
		dataset.SetFieldValue("memo",(_bstr_t)checkedit.m_memo);
		CTime latetime = DecTime(checkedit.m_timeon,checkedit.m_timeonduty);
		time.Format("%d:%d:%d",latetime.GetHour(),latetime.GetMinute(),latetime.GetSecond());
		dataset.SetFieldValue("latetime",(_bstr_t)time);
		CTime leaveearly = DecTime(checkedit.m_timeoffduty,checkedit.m_timeoff);
		time.Format("%d:%d:%d",leaveearly.GetHour(),leaveearly.GetMinute(),leaveearly.GetSecond());
		dataset.SetFieldValue("leaveearly",(_bstr_t)time);
		dataset.Save();												//保存记录集修改
		UpdateList();												//更新考勤信息列表
	}
}
```

（7）添加 OnDelete 方法，用于删除当前选择的考勤记录，实现代码如下：

```
void CCheckManage::OnDelete()
{
	if (MessageBox("是否删除此记录！ ","提示",
		MB_YESNO|MB_ICONWARNING) == IDYES)
	{
		if (m_list.GetSelectionMark() == -1)
			return;
		int id = m_list.GetItemData(m_list.GetSelectionMark());
		CADODataSet dataset;
		dataset.SetConnection(::GetConnection());
		CString str;
		str.Format("select * from tab_Check where autoid = %d",id);
		dataset.Open(str);
		dataset.Delete();
		dataset.Save();
		UpdateList();
	}
}
```



4.11　考勤汇总查询模块设计

4.11.1　考勤汇总查询模块概述

考勤汇总查询模块用于将日常录入的员工考勤信息根据时间范围和人员进行汇总查询，并显示员工的月出勤天数、迟到天数和请假天数等。考勤汇总查询模块如图 4.5 所示。

4.11.2　考勤汇总查询模块技术分析

在该模块中，汇总查询是通过 SQL 语句实现的，这个汇总查询主要通过一些 SQL 子句组成一个 SQL 汇总查询语句，这些子句分别用来获取员工工作总天数、迟到总天数、早退总天数、病假总天数和事假总天数。

在进行天数计算时是将天转换成秒，先计算所使用的总秒数，最后再根据总秒数计算出天数。考勤汇总查询的 SQL 语句如下：

```
CString str,temp,where,datestr,StartDate,EndDate;
StartDate = m_yy + "-" + m_mm + "-1";
EndDate.Format("DATEADD(month,1,'%s')",StartDate);
datestr.Format(" between '%s' and %s",StartDate,EndDate);
temp += "select emp.emp_name,ROUND(isnull(works.workday,0),2)";
temp += " workday,ROUND(isnull(lates.lateday,0),2) lateday,";
temp += " ROUND(isnull(leaveearlys.leaveearlyday,0),2) leaveearlyday,";
temp += " ROUND(isnull(bjdays.bjday,0),2) bjday,ROUND(isnull(sjdays.sjday,0),2) sjday";
```



```
temp += " from tab_Employees emp ";
temp += " left join";
temp += " (select sum(DATEDIFF(second,ontime,offtime)) / 60.0 / 60.0 / 8.0";
temp += " as workday,name   From tab_Check where checkdate %s group by name)";
temp += " works on emp.emp_name = works.name";
temp += " left join";
temp += " (select (sum(DATEPART(Hour,latetime)) * 60 * 60 + ";
temp += " sum(DATEPART(minute,latetime)) * 60 + sum(DATEPART(second,latetime)))";
temp += " /60.0 /60.0 /8.0 as lateday,name From tab_Check where checkdate";
temp += " %s group by name) lates on emp.emp_name = lates.name";
temp += " left join";
temp += " (select (sum(DATEPART(Hour,leaveearly)) * 60 * 60 + ";
temp += " sum(DATEPART(minute,leaveearly)) * 60 + sum(DATEPART(second,leaveearly)))";
temp += " /60.0 /60.0 /8.0 as leaveearlyday,name From tab_Check where ";
temp += " checkdate %s group by name) leaveearlys on emp.emp_name";
temp += " = leaveearlys.name";
temp += " left join";
temp += " (select isnull(sum(DATEDIFF(second,onleave,offleave))";
temp += " / 60.0 / 60.0 / 8.0,0) as bjday,name   From tab_Check where";
temp += " leave = '病假' and checkdate %s group by name) ";
temp += " bjdays on emp.emp_name = bjdays.name";
temp += " left join";
temp += " (select isnull(sum(DATEDIFF(second,onleave,offleave)) ";
temp += " / 60.0 / 60.0 / 8.0,0) as sjday,name   From tab_Check where ";
temp += " leave = '事假' and checkdate %s group by name) ";
temp += " sjdays on emp.emp_name = sjdays.name";
temp += " %s";
```

4.11.3　考勤汇总查询模块实现过程

本模块使用的数据表：tab_Employees、tab_Check

（1）创建一个对话框，打开对话框属性窗口，将对话框的 ID 改为 IDD_DLGCHECKSUM，将对话框标题改为“考勤汇总查询”。

（2）向对话框中添加 3 个静态文本框控件、3 个下拉列表框控件、一个按钮控件和一个列表控件，各控件的属性设置如表 4.5 所示。

表 4.5　控件属性设置

控件 ID	控 件 属 性	对 应 变 量
IDC_CYY	Type：Drop List	CComboBox m_cyy CString m_yy
IDC_CMM	Type：Drop List	CComboBox m_cmm CString m_mm
IDC_CEMP	Type：Drop List	CComboBox m_cemp CString m_emp
IDCANCEL	IDCANCEL	Caption：退出
IDC_LISTEMP	View：Icon、Single selection	CListCtrl m_list

（3）添加 UpdateList 方法，用于更新考勤汇总查询的数据，实现代码如下：

```
void CCheckSum::UpdateList()
{
	m_list.DeleteAllItems();
	this->UpdateData();
	CADODataSet dataset;
	dataset.SetConnection(::GetConnection());
	CString str,temp,where,datestr,StartDate,EndDate;
	StartDate = m_yy + "-" + m_mm + "-1";
	EndDate.Format("DATEADD(month,1,'%s')",StartDate);
	datestr.Format(" between '%s' and %s",StartDate,EndDate);
	temp += "select emp.emp_name,ROUND(isnull(works.workday,0),2)";
	temp += " workday,ROUND(isnull(lates.lateday,0),2) lateday,";
	temp += " ROUND(isnull(leaveearlys.leaveearlyday,0),2) leaveearlyday,";
	temp += " ROUND(isnull(bjdays.bjday,0),2) bjday,ROUND(isnull(sjdays.sjday,0),2) sjday";
	temp += " from tab_Employees emp ";
	temp += " left join";
	temp += " (select sum(DATEDIFF(second,ontime,offtime)) / 60.0 / 60.0 / 8.0";
	temp += " as workday,name   From tab_Check where checkdate %s group by name)";
	temp += " works on emp.emp_name = works.name";
	temp += " left join";
	temp += " (select (sum(DATEPART(Hour,latetime)) * 60 * 60 + ";
	temp += " sum(DATEPART(minute,latetime)) * 60 + sum(DATEPART(second,latetime)))";
	temp += " /60.0 /60.0 /8.0 as lateday,name From tab_Check where checkdate";
	temp += " %s group by name) lates on emp.emp_name = lates.name";
	temp += " left join";
	temp += " (select (sum(DATEPART(Hour,leaveearly)) * 60 * 60 + ";
	temp += " sum(DATEPART(minute,leaveearly)) * 60 + sum(DATEPART(second,leaveearly)))";
	temp += " /60.0 /60.0 /8.0 as leaveearlyday,name From tab_Check where ";
	temp += " checkdate %s group by name) leaveearlys on emp.emp_name";
	temp += " = leaveearlys.name";
	temp += " left join";
	temp += " (select isnull(sum(DATEDIFF(second,onleave,offleave))";
	temp += " / 60.0 / 60.0 / 8.0,0) as bjday,name   From tab_Check where";
	temp += " leave = '病假' and checkdate %s group by name) ";
	temp += " bjdays on emp.emp_name = bjdays.name";
	temp += " left join";
	temp += " (select isnull(sum(DATEDIFF(second,onleave,offleave)) ";
	temp += " / 60.0 / 60.0 / 8.0,0) as sjday,name   From tab_Check where ";
	temp += " leave = '事假' and checkdate %s group by name) ";
	temp += " sjdays on emp.emp_name = sjdays.name";
	temp += " %s";
	where.Format(" where emp.emp_name = '%s'",m_emp);
	if (m_emp == "(全部)")
		str.Format(temp,datestr,datestr,datestr,datestr,datestr,"");
	else
		str.Format(temp,datestr,datestr,datestr,datestr,datestr,where);
	dataset.Open(str,adLockUnspecified);
	for (int i = 0; i < dataset.GetRecordCount(); i++)
	{
```

```
            int n = 0;
            m_list.InsertItem(i,"");
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["emp_name"]->Value);
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["workday"]->Value);
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["lateday"]->Value);
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["leaveearlyday"]->Value);
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["bjday"]->Value);
            m_list.SetItemText(i,n++,(_bstr_t)dataset.GetFields()->Item["sjday"]->Value);
            dataset.Next();
        }
}
```

（4）向对话框中添加 OnInitDialog 方法，在对话框初始化时设置列表控件的表头和列宽度，以及汇总查询条件选择控件，实现代码如下：

```
BOOL CCheckSum::OnInitDialog()
{
        CDialog::OnInitDialog();
        int i = 0;
        m_list.InsertColumn(i,"人员姓名");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"工作总天数");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"迟到总天数");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"早退总天数");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"病假总天数");
        m_list.SetColumnWidth(i++,100);
        m_list.InsertColumn(i,"事假总天数");
        m_list.SetColumnWidth(i++,100);
        m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
        int curyear,curmonth;
        CTime time(CTime::GetCurrentTime());
        curyear = time.GetYear();
        curmonth = time.GetMonth();
        char value[10];
        for (int y = 2000; y < 2100;y++)
        {
❶               _itoa(y,value,10);
                m_cyy.InsertString(y-2000,value);
        }
❷       m_cyy.SetCurSel(curyear-2000);
        for (int n = 1; n<=12;n++)
        {
                _itoa(n,value,10);
                m_cmm.InsertString(n-1,value);
        }
        m_cmm.SetCurSel(curmonth-1);
        CADODataSet dataset;
```

```
        dataset.SetConnection(::GetConnection());
        dataset.Open("Select * From tab_Employees");
        m_cemp.InsertString(0,"(全部)");
        for (int index = 1; index < dataset.GetRecordCount(); index++)
        {
                m_cemp.InsertString(index,(_bstr_t)dataset.GetFields()->Item["emp_name"]->Value);
                dataset.Next();
        }
        m_cemp.SetCurSel(0);
        UpdateList();
        return TRUE;
}
```

代码贴士

❶ _itoa 函数：用于将整型数据转换为字符串类型。

❷ SetCurSel 方法：用于设置组合框中的选中项。

4.12　开发技巧与难点分析

4.12.1　调用动态链接库设计界面

为了使人事考勤管理系统在界面上更加美观，笔者重绘了程序界面，并且将其封装成了动态链接库，使读者可以方便地使用。下面就来看一下如何调用动态链接库美化界面，步骤如下：

（1）将资源包中提供的 SkinHook.h、SkinLib.dll 和 SkinLib.lib 文件复制到程序根目录下，如图 4.22 所示。

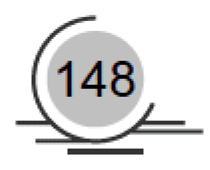
（2）在工作区窗口选择 FileView 选项卡，在 Header Files 节点处右击，在弹出的快捷菜单中选择 Add Files to Folder 命令，在弹出的对话框中找到 SkinHook.h 文件并导入。

（3）在 Person.cpp 文件中引用 SkinHook.h 文件，并在 InitInstance 方法中调用动态链接库中的 LoadSkin 函数进行界面的绘制。

图 4.22　复制动态链接库文件

4.12.2　主窗口的界面显示

在开发本系统的主窗口时，为了使程序看起来更加美观，在程序的背景部分绘制了一幅位图，但是在程序进行最大化时，却出现了问题，程序的背景位图没有被重绘，导致左上角的部分显示一个小



图，这该如何解决呢？

可以先捕获最大化消息，然后调用 Invalidate 函数进行刷新。可以在主窗口的 OnSize 消息处理函数中捕获最大化消息，代码如下：

```
void CPersonDlg::OnSize(UINT nType, int cx, int cy)
{
	CDialog::OnSize(nType, cx, cy);
	if (nType == SIZE_MAXIMIZED)                        //捕获最大化消息
	{
		Invalidate();                                   //刷新
	}
	else if (nType == SIZE_RESTORED)                    //捕获还原消息
	{
		Invalidate();                                   //刷新
	}
}
```

通过上述代码就可以解决调整主窗口大小时背景位图显示不正确的问题。

4.13　项目文件清单

人事考勤管理系统项目文件清单如表 4.6 所示。

表 4.6　项目文件清单

文件名称	文件类型	文件描述	文件名称	文件类型	文件描述
ADO.cpp	源文件	连接数据库	DeptManage.cpp	源文件	部门管理
CheckEdit.cpp	源文件	考勤校订	LoginDialog.cpp	源文件	用户登录
CheckManage.cpp	源文件	考勤管理	PassWordEdit.cpp	源文件	分类列表控制
CheckSum.cpp	源文件	考勤汇总	Person.cpp	源文件	人事考勤文件
DeptEdit.cpp	源文件	部门校订	Person.rc	资源文件	系统资源文件
PersonDlg.cpp	源文件	人事资源框	PersonEdit.cpp	源文件	人事校订
PersonManage.cpp	源文件	人事管理	UserEdit.cpp	源文件	用户编辑
UserManage.cpp	源文件	用户管理			

4.14　本 章 总 结

本章通过使用 SQL Server 2014 数据库介绍了如何开发人事考勤管理系统。通过本章的学习，读者可以更好地掌握 SQL Server 2014 数据库开发技术，增强对数据库管理系统开发流程的了解，可以向独立开发软件的目标迈出一大步。

第 5 章

商品采购管理系统

（Visual C++ 6.0+SQL Server 2014 实现）

全球经济一体化步伐的加快，要求企业快速、全面发展，以适应整个市场的发展变化。要使企业全面发展，不仅要求企业不断地改善经营状况，更重要的是要不断地提高企业生产经营活动中“供、销、存”各个环节的管理水平，因此商品采购管理水平的提高在整个管理过程中变得日益重要。

通过学习本章，读者可以学到：

- **使用 SQL Server 2014 数据库**
- **通过 SQL 语句对数据库进行操作**
- **备份和还原数据库**
- **数据库封装类**



配置用说明

5.1　开 发 背 景

通过商品采购管理系统对企业的商品采购进行管理，满足了企业对商品采购及时、采购数量预算准确、采购商品质量有保障等要求，为企业的经营生产做好前提准备工作，从而提高了企业“供、销、存”管理的整体水平。

5.2　需 求 分 析

商品采购管理系统根据工业企业和商品流通企业采购业务管理及采购成本核算的实际需要，对采购订单、采购到货处理及入库状况进行全程管理，为采购部门和财务部门提供准确、及时的信息，并辅助管理决策。

本系统完成后，能够输入或修改商品和供应商的基本资料，能方便对采购业务和交货信息进行维护，能对商品采购信息进行查询、交货追踪和统计。

5.3　系 统 设 计



5.3.1　系统目标

商品采购管理系统将实现如下目标。

☑　减少前台服务人员的数量，减少经营者的人员开销。

☑　提高操作速度，提高顾客的满意程度。

☑　使经营者能够查询一些历史数据。

5.3.2　系统功能结构

商品采购管理系统的功能结构如下。

☑　基础信息管理：在基础信息管理模块中需要实现对部门信息、员工信息、系统功能、角色、操作员、商品信息和库存信息的管理功能。

☑　采购管理：在采购管理模块中需要实现采购申请、采购入库和入库退货功能。

☑　采购查询：在采购查询模块中需要实现对采购申请信息、采购入库信息和入库退货信息的查询功能。

系统结构图如图 5.1 所示。

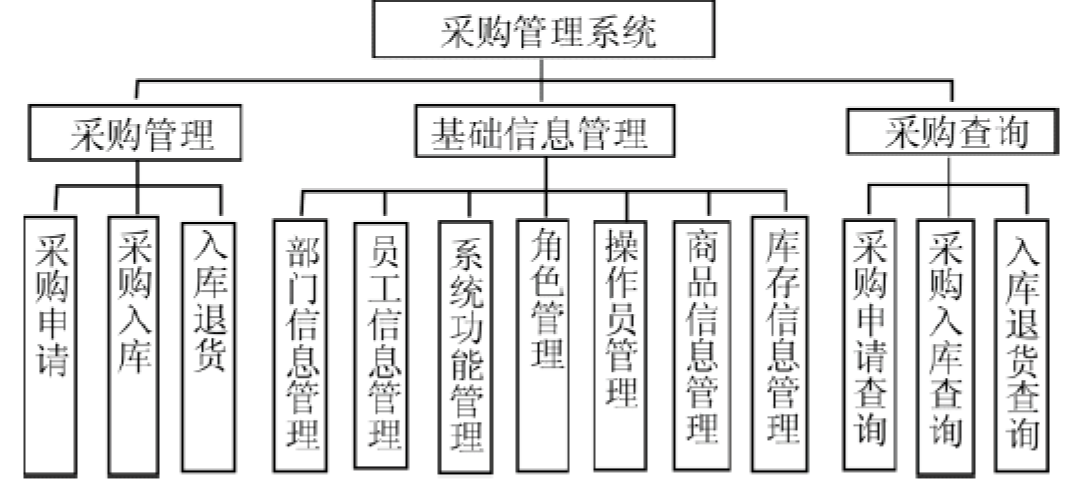
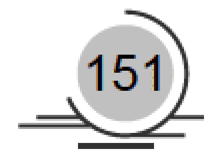


图 5.1　采购管理系统结构图



5.3.3　系统　览

商品采购管理系统主要由采购申请模块组成，模块效果预览图如图 5.2 所示。

5.3.4　业务流程图

商品采购管理系统业务流程图如图 5.3 所示。

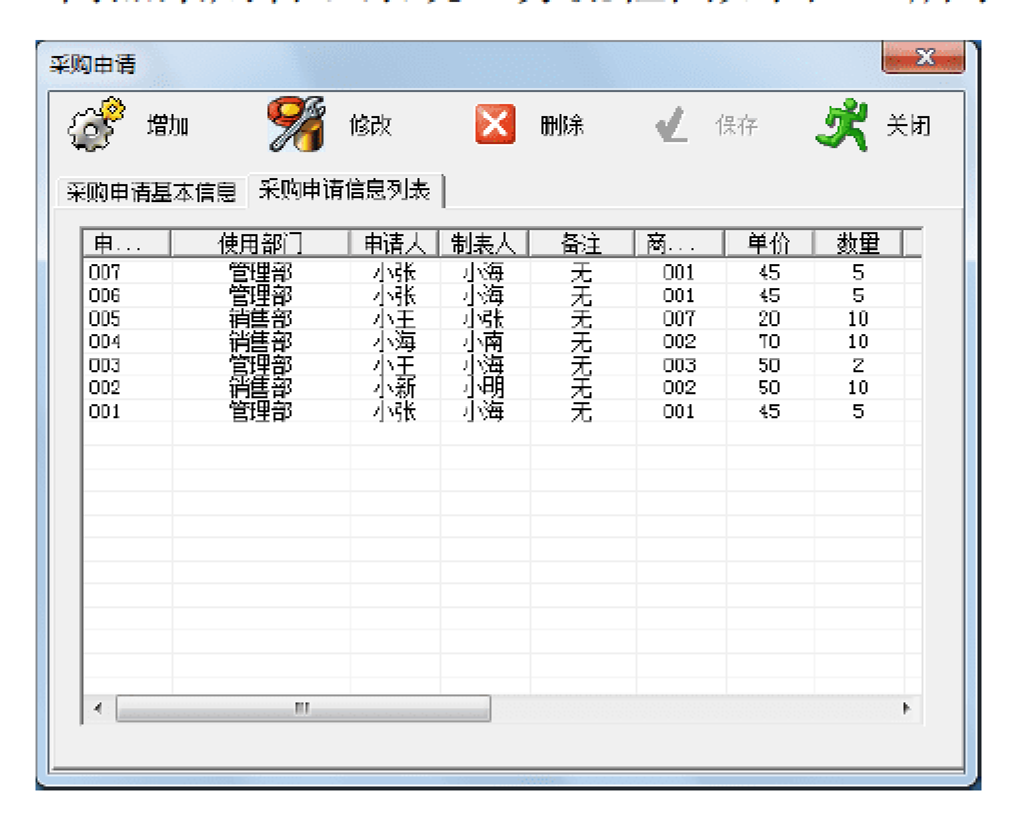


图 5.2　采购申请模块的运行效果

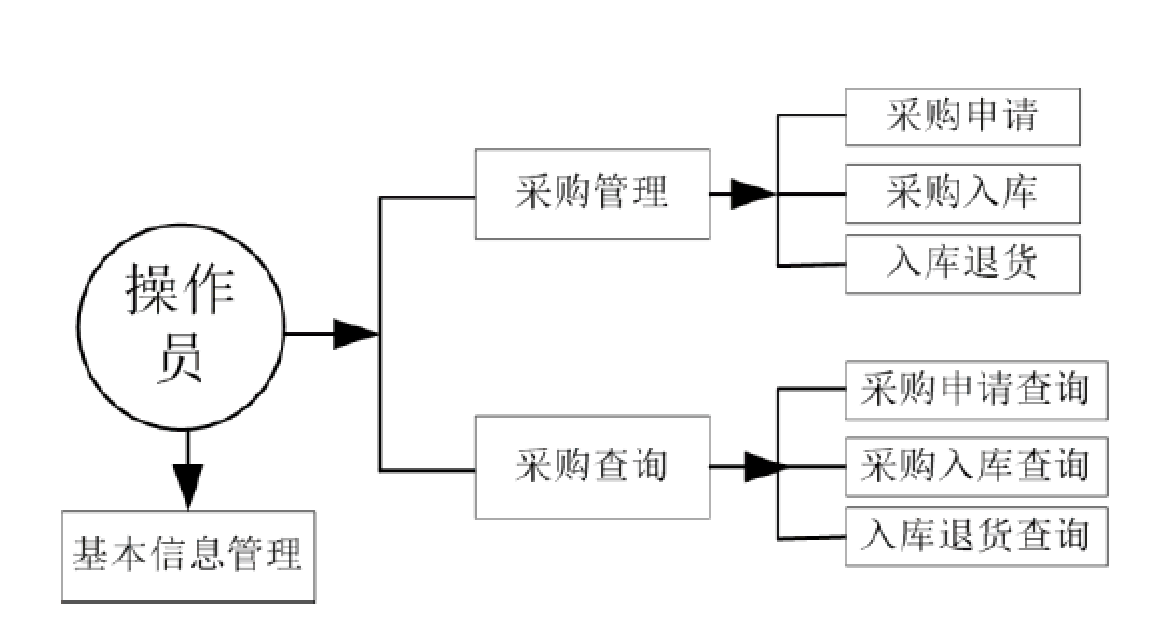


图 5.3　商品采购系统业务流程图

5.3.5　数据库设计

1．数据库概要说明

在商品采购管理系统中，采用的是 SQL Server 2014 数据库，用来存储供应商信息、员工信息、角色信息、角色功能、部门信息、系统功能信息、操作员信息、采购申请、采购申请明细、订单信息、采购入库、商品信息、商品库存、库存信息、入库退货主和入库存退货明细等。这里将数据库命名为 StockManage，其中包含了 16 张数据表，用于存储不同的信息，如图 5.4 所示。

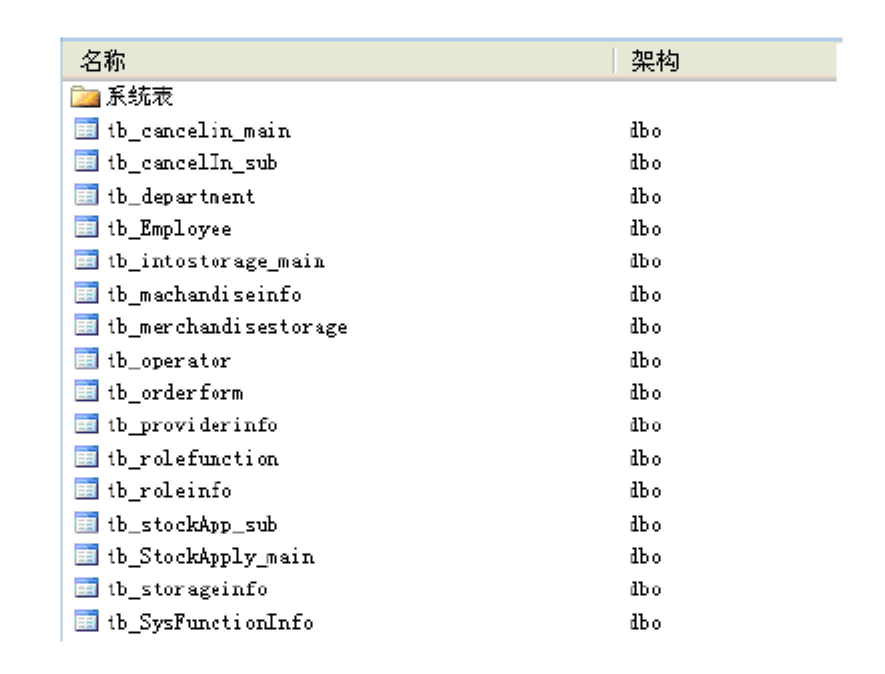
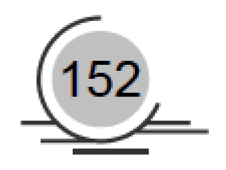


图 5.4　“采购管理系统”数据库结构

2．数据库　　结构设计

采购管理系统共使用了 16 张数据表，分别为供应商信息表（tb_providerinfo）、员工信息表（tb_Employee）、角色信息表（tb_roleinfo）、角色功能表（tb_rolefunction）、部门信息表（tb_department）、系统功能信息表（tb_SysfunctionInfo）、操作员信息表（tb_operator）、采购申请主表（tb_StockApply_main）、采购申请明细表（tb_stockApp_sub）、订单信息表（tb_orderform）、采购入库表（tb_intostorage_main）、商品信息表（tb_machandiseinfo）、商品库存表（tb_merchandisestorage）、库存信息表（tb_storageinfo）和入库退货主表（tb_cancelin_main）和入库存退货明细表（tb_cancelIn_sub）。由于篇幅关系，在此只列



出使用较多的数据表结构，如表 5.1～表 5.6 所示，其他表的结构请参考资源包。

表 5.1　商品信息表（tb_machandiseinfo）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	认　值
ID	varchar	是			
name	varchar				名称
spec	varchar			是	规格
shortmane	varchar				助记码
defaultprice	money				默认价格
manufacturer	varchar			是	厂家
memo	varchar			是	备注

表 5.2　入库　货主表（tb_cancelin_main）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	描
CancelID	varchar	是			退货单号
ProviderID	varchar				供应商编号
Operator	varchar				操作员
principal	varchar				负责人
Rebate	float				折扣
SumTotal	money				总计
Paymoney	money				应返余额
Factmoney	money				实返余额
CancelTime	Datetime				返货时间

表 5.3　　购申请主表（tb_StockApply_main）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	描
AppID	varchar	是			申请单号
Department	varchar				使用部门
proposer	varchar				申请人
billmaker	varchar				制单人
memo	varchar			是	备注
AppDate	datetime				日期

表 5.4　订单信息表（tb_orderform）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	描
orderID	varchar	是			订单号
billmaker	varchar				制单人
ordertime	datetime				日期
InStored	smallint				是否入库
providerID	varchar				供应商
sumtotal	money				总计
rebate	money				折扣
paymoney	money				应付余额

表 5.5　供应商信息表（tb_providerinfo）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	描
供应商编号	varchar	是			供应商编号
供应商名称	varchar				供应商名称
法人	varchar				法人
负责人	varchar			是	负责人
联系电话	varchar			是	联系电话
详细地址	varchar			是	详细地址
网址	varchar			是	网址
邮箱	varchar			是	邮箱

表 5.6　购申请明细表（tb_stockApp_sub）

字 段 名 称	字 段 类 型	主	外	是 否 为 空	描
appID	varchar				申请单号
merchandiseID	varchar				商品编号
unitprice	money			是	单价
[sum]	float				数量
Paymoney	money			是	余额
Rebate	float			是	折扣
BestTime	datetime			是	建议采购日期
purpose	varchar			是	用途
providerid	varchar			是	供应商
orderform	varchar			是	订单号
StockName	char			是	库存名称



5.4　数据库封装类说明

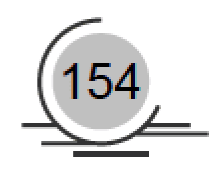
5.4.1　数据库封装类概

商品采购管理系统采用 ADO 技术操作数据库。为了方便，笔者将程序中需要用到的 ADO 对象进行了封装。

5.4.2　数据库封装类步

（1）在工作区的类视图的 MerchandiseSell classes 节点上右击，在弹出的快捷菜单中选择 New Class 命令，将弹出 New Class 对话框。在该对话框的 Class type 下拉列表框中选择 Generic Class 选项，在 Name 文本框中输入“CDatabase”，如图 5.5 所示。

（2）单击 OK 按钮，生成自定义类 CDatabase，如图 5.6 所示。





（3）选择此类，右击，在弹出的快捷菜单中，Add Member Function 命令用于添加成员函数，Add Member Variable 命令用于添加成员变量，如图 5.7 所示。

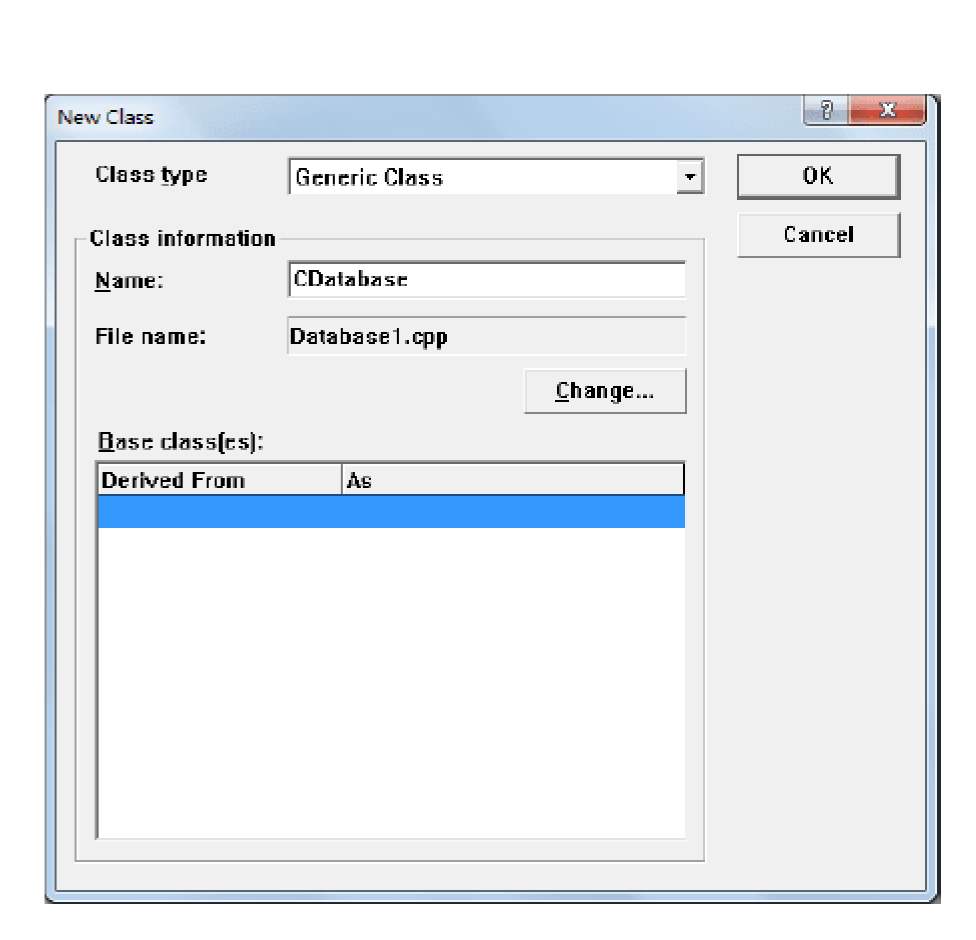


图 5.5　New Class 对话框

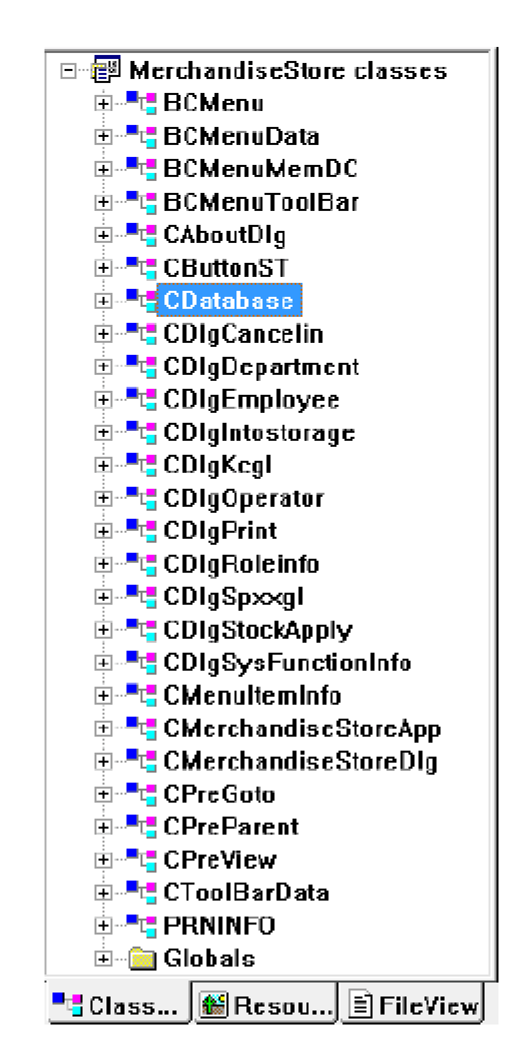


图 5.6　生成自定义类 CDatabase

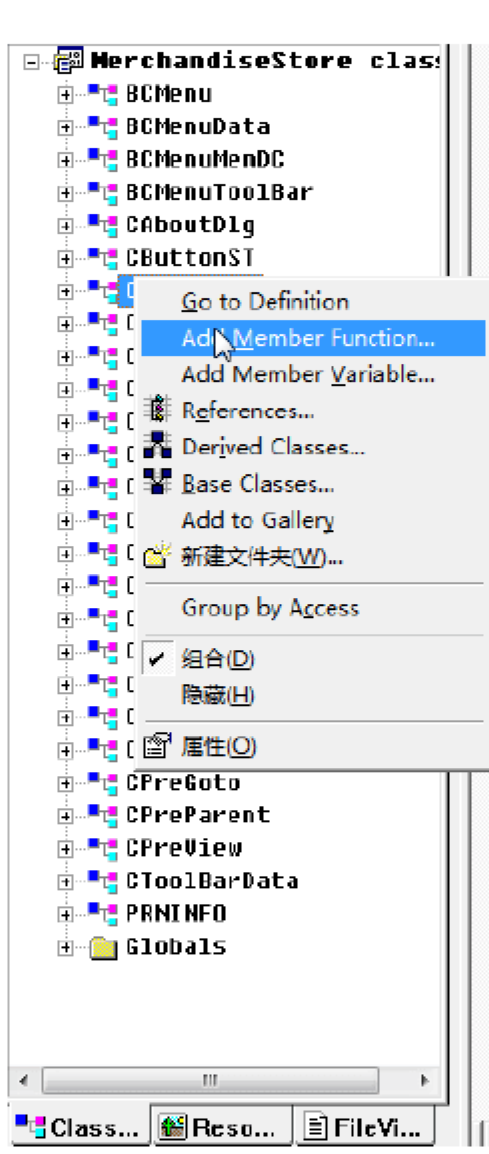


图 5.7　添加成员函数

（4）添加数据库操作句柄为私有变量，如图 5.8 所示。

（5）添加数据库初始化函数，如图 5.9 所示。



图 5.8　添加数据库操作句柄

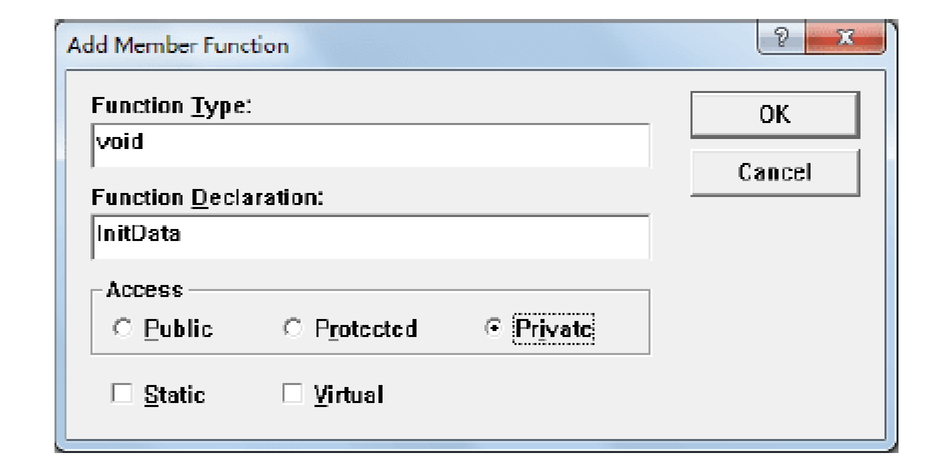


图 5.9　添加数据库初始化函数



所有函数添加以此类推。

5.4.3　数据库封装类实现　程

InitData 成员函数用于初始化数据库连接，返回 1 为连接成功，返回 0 为失败。

```
int CDatabase::InitData()
{
	char m_szConnect[512];
	char m_szTmp[1024]="";
```



```
        char m_szHost[20], m_szUser[20], m_szPwd[20], m_szDef[20];
        GetPrivateProfileString("数据库", "主机名", NULL, m_szHost, sizeof(m_szHost), IniFile);
        GetPrivateProfileString("数据库", "用户名", NULL, m_szUser, sizeof(m_szUser), IniFile);
        GetPrivateProfileString("数据库", "密码", NULL, m_szPwd, sizeof(m_szPwd), IniFile);
        GetPrivateProfileString("数据库", "   认库", NULL, m_szDef, sizeof(m_szDef), IniFile);
        try{   //   接 XdData
                HRESULT hr = m_Connection.CreateInstance(__uuidof(Connection));
                sprintf(m_szConnect,"provider = sqloledb;server=%s;database=%s;", m_szHost, m_szDef);
        hr=m_Connection->Open(_bstr_t(m_szConnect),_bstr_t(m_szUser),_bstr_t(m_szPwd),-1);
                sprintf(m_szTmp, "数据库   接成功!");
        }
        catch(_com_error & e)
        {
                sprintf(m_szTmp, "数据库打开失败，   误原因：%s\n",LPCTSTR(e.Description()));
                return 0;
        }
        return 1;
}
```

IsVerifyUser 成员函数用于登录校验管理员身份。

```
int CDatabase::IsVerifyUser(char *m_szUser, char *m_szPwd, char *m_szLevel)
{
        _variant_t v(0L);
        _RecordsetPtr m_Rsp;
        char m_szSql[512];
        sprintf(m_szSql, "select * from tb_operator where name = '%s' and password = '%s'", m_szUser,
m_szPwd);
        try{
                m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
                if(!m_Rsp->GetadoEOF())
                {
                        v = m_Rsp->GetCollect("level");
                        if(atoi(_bstr_t(v)) == 0)
                        {
                                strcpy(m_szLevel, "系统管理员");                      //系统
                        }
                        else
                        {
                                strcpy(m_szLevel, "普   管理员");                     //普
                        }
                        return 1;
                }
        }
        catch(_com_error & e)
        {
                char m_szTmp[1024];
                sprintf(m_szTmp, "执行==>%s<==，数据库操作失败，   误原因：%s\n",m_szSql, LPCTSTR(e.Description()));
                return -1;
```

```
    }
    return 0;
}
```

DeleteDataWhere 成员函数用于根据条件删除数据库中的记录。

```
void CDatabase::DeleteDataWhere(int m_nIndex, char *Fond)
{
    char m_szSql[512];
    _variant_t v(0L);
    switch(m_nIndex)
    {
    case OPT:
        {
            sprintf(m_szSql, "delete from tb_operator where name = '%s'", Fond);
            break;
        }
    case SPXXT:
        {
            sprintf(m_szSql, "delete from tb_merchandiseinfo where id = '%s'", Fond);
            break;
        }
    case GYSXX:
        {
            sprintf(m_szSql, "delete from tb_providerinfo where provider = '%s'", Fond);
            break;
        }
    case KHXX:
        {
            sprintf(m_szSql, "delete from tb_customerinfo where name = '%s'", Fond);
            break;
        }
    case KCGL:
        {
            sprintf(m_szSql, "delete from tb_merchandisestorage where merchandiseID = '%s'", Fond);
            break;
        }
    case SPRK:
        {
            sprintf(m_szSql, "delete from tb_instore_main where ID = '%s'", Fond);
            break;
        }
    case RKTH:
        {
            sprintf(m_szSql, "delete from tb_cancelinstock_main where CancelID = '%s'", Fond);
            break;
        }
    case XSTH:
        {
```

```
                sprintf(m_szSql, "delete from tb_cancelsell_main where CancelID = '%s'", Fond);
                break;
            }
        case SPXS:
            {
                sprintf(m_szSql, "delete from tb_sell_main where CancelID = '%s'", Fond);
                break;
            }
        case GYSJK:
            {
                sprintf(m_szSql, "delete from tb_providerpay where PayID = '%s'", Fond);
                break;
            }
        case KHJK:
            {
                sprintf(m_szSql, "delete from tb_customerpay where PayID = '%s'", Fond);
                break;
            }
        }
        try{
            if(m_nIndex==SPRK)
            {
                m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
                sprintf(m_szSql, "delete from tb_instock_sub where instockid = '%s'", Fond);
            }
            else    if(m_nIndex==RKTH)
            {
                m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
                sprintf(m_szSql, "delete from tb_cancelinstock_sub where CancelID = '%s'", Fond);
            }
            else    if(m_nIndex==XSTH)
            {
                m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
                sprintf(m_szSql, "delete from tb_cancelsell_sub where CancelID = '%s'", Fond);
            }
            else    if(m_nIndex==SPXS)
            {
                m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
                sprintf(m_szSql, "delete from tb_sell_sub where CancelID = '%s'", Fond);
            }
            m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        }
        catch(_com_error & e)
        {
            char m_szTmp[1024];
            sprintf(m_szTmp, "执行==>%s<==，数据库操作失败，    误原因：%s\n",m_szSql, LPCTSTR(e.Description()));
        }
    }
```

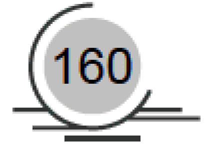
下面的代码用于将所需数据显示在 LIST 列表中，与名为 List…ToCtrl 的函数用途相同，不同之处在于操作的数据表不同。

```
void CDatabase::ListCancelInStockToCtrl(CListCtrl *m_hListCtrl)
{
    m_hListCtrl->DeleteAllItems();
    _variant_t v(0L);
    _RecordsetPtr m_Rsp;
    char m_szSql[512];
    sprintf(m_szSql, "select * from tb_cancelinstock_main a, tb_cancelinstock_sub b where a.CancelID =
b.CancelID");
    try{
        m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        while(!m_Rsp->GetadoEOF())
        {
            m_hListCtrl->InsertItem(0, "");
            v = m_Rsp->GetCollect("CancelID");
            m_hListCtrl->SetItemText(0, 0, _bstr_t(v));
            v = m_Rsp->GetCollect("Provider");
            m_hListCtrl->SetItemText(0, 1, _bstr_t(v));
            v = m_Rsp->GetCollect("operator");
            m_hListCtrl->SetItemText(0, 2, _bstr_t(v));
            v = m_Rsp->GetCollect("rebate");
            m_hListCtrl->SetItemText(0, 3, _bstr_t(v));
            v = m_Rsp->GetCollect("sumtotal");
            m_hListCtrl->SetItemText(0, 4, _bstr_t(v));
            v = m_Rsp->GetCollect("paymoney");
            m_hListCtrl->SetItemText(0, 5, _bstr_t(v));
            v = m_Rsp->GetCollect("factmoney");
            m_hListCtrl->SetItemText(0, 6, _bstr_t(v));
            v = m_Rsp->GetCollect("stockname");
            m_hListCtrl->SetItemText(0, 7, _bstr_t(v));
            v = m_Rsp->GetCollect("merchandiseID");
            m_hListCtrl->SetItemText(0, 8, _bstr_t(v));
            v = m_Rsp->GetCollect("unitPrice");
            m_hListCtrl->SetItemText(0, 9, _bstr_t(v));
            v = m_Rsp->GetCollect("numbers");
            m_hListCtrl->SetItemText(0, 10, _bstr_t(v));
            v = m_Rsp->GetCollect("paymoney");
            m_hListCtrl->SetItemText(0, 11, _bstr_t(v));
            v = m_Rsp->GetCollect("intime");
            m_hListCtrl->SetItemText(0, 12, _bstr_t(v));
            m_Rsp->MoveNext();
        }
    }
    catch(_com_error & e)
    {
        char m_szTmp[1024];
```

```
            sprintf(m_szTmp, "执行==>%s<==，数据库操作失败，  误原因：%s\n",m_szSql, LPCTSTR(e.Description()));
        }
    }
```

下面的代码用于将所需数据显示在对应的编辑框中，与名为 Edit…ToCtrl 的函数用途相同，不同之处在于操作的数据表不同。

```
void CDatabase::EditCancelInStockToCtrl(char *CancelID, CEdit *m_hEditCancelID, CEdit *m_hEditUnitPrice,
CEdit *m_hEditSumTotal, CEdit *m_hEditStockName, CEdit *m_hEditRebate, CEdit *m_hEditPayMoney, CEdit
*m_hEditOperator, CEdit *m_hEditNumbers, CEdit *m_hEditMerchandiseID, CEdit *m_hEditFactMoney, CEdit
*m_hEditProvider)
{
    _variant_t v(0L);
    _RecordsetPtr m_Rsp, m_Rsp1;
    char m_szSql[512];
    sprintf(m_szSql, "select * from tb_cancelinstock_main where CancelID = '%s'", CancelID);
    try{
        m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        while(!m_Rsp->GetadoEOF())
        {
            v = m_Rsp->GetCollect("CancelID");
            m_hEditCancelID->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("Provider");
            m_hEditProvider->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("operator");
            m_hEditOperator->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("rebate");
            m_hEditRebate->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("sumtotal");
            m_hEditSumTotal->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("paymoney");
            m_hEditPayMoney->SetWindowText(_bstr_t(v));
            v = m_Rsp->GetCollect("factmoney");
            m_hEditFactMoney->SetWindowText(_bstr_t(v));
            sprintf(m_szSql, "select * from tb_cancelinstock_sub where SellID = '%s'", CancelID);
            m_Rsp1 = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
            if(!m_Rsp1->GetadoEOF())
            {
                v = m_Rsp1->GetCollect("merchandiseID");
                m_hEditMerchandiseID->SetWindowText(_bstr_t(v));
                v = m_Rsp1->GetCollect("unitPrice");
                m_hEditUnitPrice->SetWindowText(_bstr_t(v));
                v = m_Rsp1->GetCollect("numbers");
                m_hEditNumbers->SetWindowText(_bstr_t(v));
                v = m_Rsp1->GetCollect("stockname");
                m_hEditStockName->SetWindowText(_bstr_t(v));
            }
            m_Rsp->MoveNext();
        }
```



```
	}
	catch(_com_error & e)
	{
		char m_szTmp[1024];
		sprintf(m_szTmp, "执行==>%s<==，数据库操作失败，	误原因：%s\n",m_szSql, LPCTSTR(e.Description()));
	}
}
```

下面的代码用于将所需数据更新，与名为 Update…ToCtrl 的函数用途相同，不同之处在于操作的数据表不同。如果查询数据存在则更新数据，不存在则插入数据。

```
void CDatabase::UpdateKhxxData(char *name, char *principal, char *phone, char *addr, char *web, char *e_mail)
{
	char m_szSql[512];
	_variant_t v(0L);
	sprintf(m_szSql, "select * from tb_customerinfo where name = '%s'", name);
	_RecordsetPtr m_Rsp;
	try{
		m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
		if(!m_Rsp->GetadoEOF())
		{//存在数据，更新
			sprintf(m_szSql, "update tb_customerinfo set principal = '%s', \phone = '%s', addr = %s, web = '%s',
e_mail = '%s' \where name = '%s'", principal, phone, addr, web, e_mail, name);
			m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
		}
		else
		{//不存在数据，增加
			sprintf(m_szSql, "insert into tb_customerinfo (principal, phone, addr, web, e_mail, name) \
				values ('%s', '%s', '%s', '%s', %s, '%s')" , principal, phone, addr, web, e_mail, name);
			m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
		}
	}
	catch(_com_error & e)
	{
		char m_szTmp[1024];
		sprintf(m_szTmp, "执行==>%s<==，数据库操作失败，	误原因：%s\n",m_szSql, LPCTSTR(e.Description()));
	}
}
```

ListDepartmentToCtrl 函数的代码如下：

```
void CDatabase::ListDepartmentToCtrl(CListCtrl *m_hListCtrl)
{
	m_hListCtrl->DeleteAllItems();

	_variant_t v(0L);
	_RecordsetPtr m_Rsp;
	char m_szSql[512];
	sprintf(m_szSql, "select * from tb_department");
```

```
	try{

		m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
		while(!m_Rsp->GetadoEOF())
		{
			m_hListCtrl->InsertItem(0, "");

			v = m_Rsp->GetCollect("departmentname");
			m_hListCtrl->SetItemText(0, 0, _bstr_t(v));

			m_Rsp->MoveNext();
		}
	}
	catch(_com_error & e)
	{
		char m_szTmp[1024];
		sprintf(m_szTmp, "执行==>%s<==, 数据库操作失败, 误原因: %s\n",m_szSql, LPCTSTR
(e.Description()));
	}

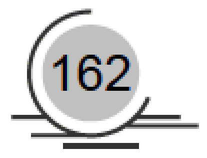
}
```

UpdateDepartmentData 函数的代码如下：

```
void CDatabase::UpdateDepartmentData(char *m_szName)
{
	char m_szSql[512];
	_variant_t v(0L);
	try{
		sprintf(m_szSql, "insert into tb_department (departmentname) values ('%s')", m_szName);
		m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
	}
	catch(_com_error & e)
	{
		char m_szTmp[1024];
		sprintf(m_szTmp, "执行==>%s<==, 数据库操作失败, 误原因: %s\n",m_szSql, LPCTSTR
(e.Description()));
	}
}
```

UpdateStockApplyData 函数的代码如下：

```
void CDatabase::UpdateStockApplyData(char *Department, char *unitprice,char *sum,char *StockName,char
*Rebate,char *Purpose,char *providerid,char *orderform,char *proposer,char *Paymoney,char *merchandiseID,
char *memo,char *Billmaker,char *Appid)
{
	char m_szSql[512];
	_variant_t v(0L);
```



```
    try{
        sprintf(m_szSql, "select * from tb_StockApply_main where appid = '%s'", Appid);
        _RecordsetPtr m_Rsp;
        m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        if(!m_Rsp->GetadoEOF())
        {//存在，更新
            sprintf(m_szSql, "update tb_StockApply_main set Department = '%s', \
                proposer = '%s', billmaker = '%s', memo = '%s' where appid='%s'",
                Department, proposer,Billmaker,memo,Appid);
            m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        }
        else
        {//不存在数据，增加
            sprintf(m_szSql, "insert into tb_StockApply_main (AppID,Department, proposer,billmaker,memo,
AppDate) \
                    values ('%s','%s','%s','%s','%s',GetDate())", Appid,Department, proposer,Billmaker,
memo);
            m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        }
        sprintf(m_szSql, "select * from tb_orderform where orderiD = '%s'", orderform);
        m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        if(!m_Rsp->GetadoEOF())
        {//存在，更新
            sprintf(m_szSql, "update tb_orderform set billmaker = '%s', \
                InStored = %s, providerID = '%s', sumtotal = %s, rebate = '%s', paymoney = %s \
                where orderiD='%s'",
                Billmaker, 0,providerid,sum,Rebate,Paymoney,orderform);
        }
        else
        {//不存在数据，增加
            sprintf(m_szSql, "insert into tb_orderform (billmaker, InStored,providerID,sumtotal,\
                rebate,paymoney,orderiD,ordertime) values ('%s',%d,'%s',%s,%s,%s,'%s',GetDate())",
                    Billmaker, 0,providerid,sum,Rebate,Paymoney,orderform);
            m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        }
        sprintf(m_szSql, "select * from tb_stockApp_sub where appid = '%s'", Appid);
        m_Rsp = m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
        if(!m_Rsp->GetadoEOF())
        {//存在，更新
            sprintf(m_szSql, "update tb_stockApp_sub set merchandiseID = '%s', \
                unitprice = %f, [sum] = %s, Rebate = %s, purpose = '%s', providerid = '%s', orderform =
'%s', StockName = '%s',Paymoney='%s' where appid='%s'",
                merchandiseID,
unitprice,sum,Rebate,Purpose,providerid,orderform,StockName,Paymoney,Appid);
        }
        else
        {//不存在数据，增加
            sprintf(m_szSql, "insert into tb_stockApp_sub (merchandiseID, unitprice,sum,Rebate,Purpose,
providerid,orderform,StockName,Appid,Paymoney,BestTime) \
```

```
                              values ('%s',%s,%s,%s,'%s','%s','%s','%s','%s',%s,GetDate())",
                              merchandiseID,
unitprice,sum,Rebate,Purpose,providerid,orderform,StockName,Appid,Paymoney);
                        m_Connection->Execute(_bstr_t(m_szSql), &v, adCmdText);
            }
      }
      catch(_com_error & e)
      {
            char m_szTmp[1024];
            sprintf(m_szTmp, "执行==>%s<==, 数据库操作失败, 误原因: %s\n",m_szSql, LPCTSTR
(e.Description()));
      }
}
```


视频讲解

5.5　主窗体设计

5.5.1　主窗体概

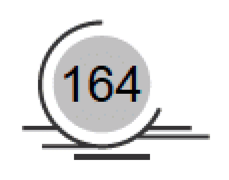
菜单是应用程序经常使用的界面元素，对应应用程序的一项功能，选择菜单项将会执行已定义的操作。主窗体运行效果如图 5.10 所示。



图 5.10　主窗体运行效果

5.5.2　主窗体实现　程

背景画面使用 Control 工具箱中的 Picture 控件实现，该控件支持 BMP 位图模式。设计背景画面的步骤如下：



（1）在 Control 工具箱中选择 Picture 控件，在窗体上添加一个 Picture 控件。

（2）在 Picture 控件上右击，在弹出的快捷菜单中选择 Properties 命令，打开 Picture Properties 对话框，在该对话框中选择 General 选项卡，如图 5.11 所示。

（3）在 Type 下拉列表框中选择 Bitmap 选项，在 Image 下拉列表框中选择 IDB_BITMAP1 选项，如图 5.12 所示。

（4）在主窗口中，所有菜单都是通过消息映射机制完成的，可通过类向导菜单来完成映射，如图 5.13 所示。

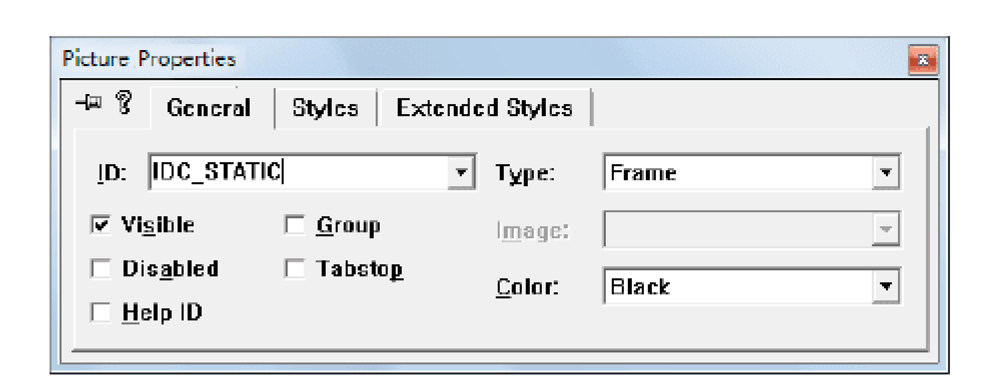


图 5.11　General 选项卡

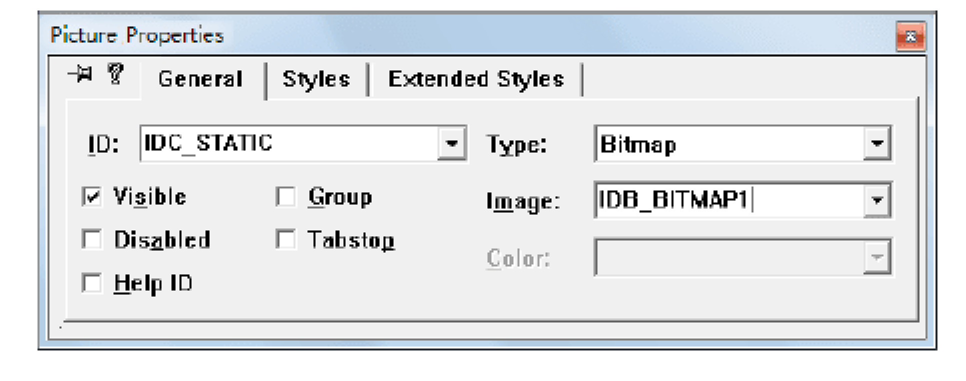


图 5.12　设置 Picture 控件背景

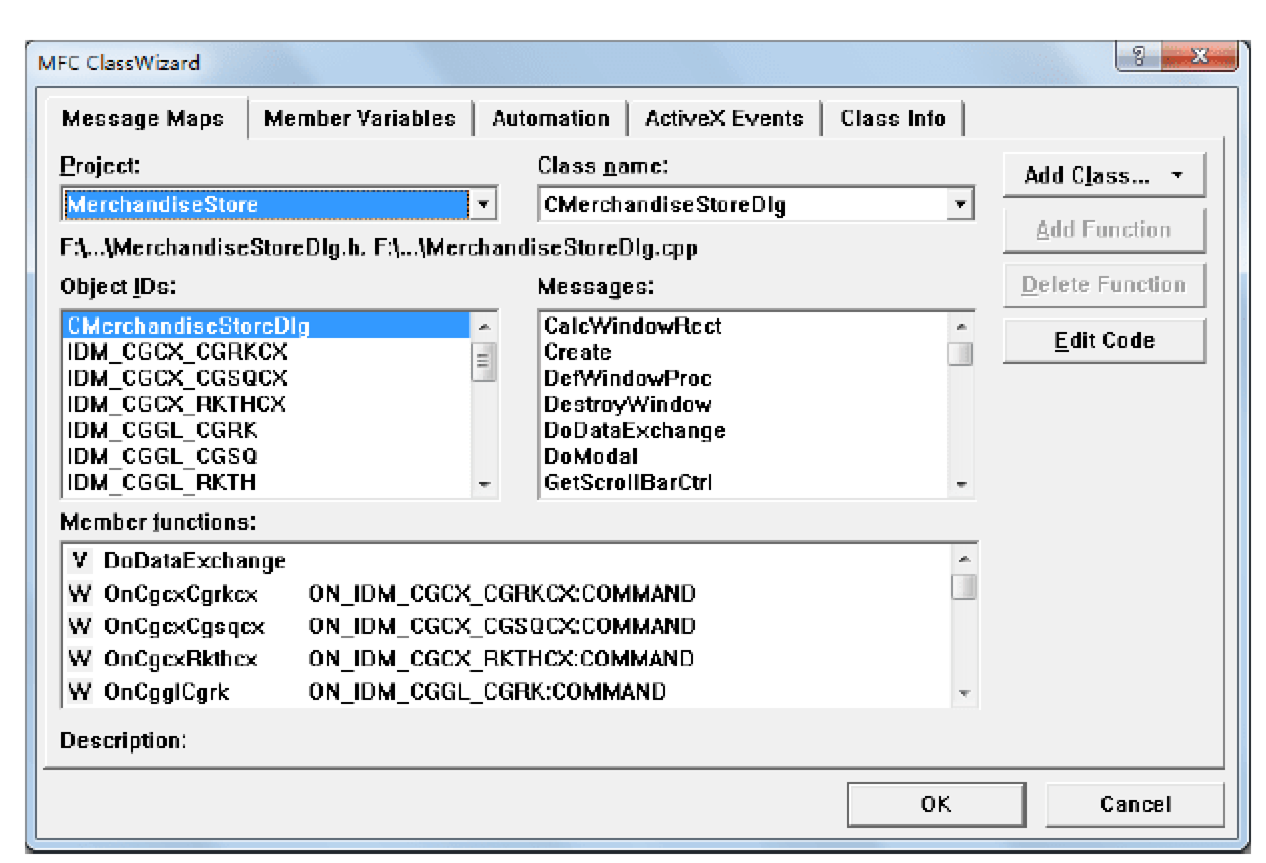
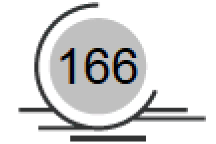


图 5.13　设置消息映射机制

映射代码及相关函数如下：

```
ON_COMMAND(IDM_JCXXGL_BMXXGL, OnJcxxglBmxxgl)
ON_COMMAND(IDM_JCXXGL_YGXXGL, OnJcxxglYgxxgl)
ON_COMMAND(IDM_JCXXGL_XTGNGL, OnJcxxglXtgngl)
ON_COMMAND(IDM_JCXXGL_JSGL, OnJcxxglJsgl)
ON_COMMAND(IDM_JCXXGL_CZYGL, OnJcxxglCzygl)
ON_COMMAND(IDM_JCXXGL_SPXXGL, OnJcxxglSpxxgl)
ON_COMMAND(IDM_JCXXGL_KCXXGL, OnJcxxglKcxxgl)
ON_COMMAND(IDM_CGGL_CGSQ, OnCgglCgsq)
ON_COMMAND(IDM_CGGL_CGRK, OnCgglCgrk)
ON_COMMAND(IDM_CGGL_RKTH, OnCgglRkth)
ON_COMMAND(IDM_CGCX_CGSQCX, OnCgcxCgsqcx)
ON_COMMAND(IDM_CGCX_CGRKCX, OnCgcxCgrkcx)
ON_COMMAND(IDM_CGCX_RKTHCX, OnCgcxRkthcx)
ON_COMMAND(IDM_WLZGL_GYSJK, OnWlzglGysjk)
void CMerchandiseStoreDlg::OnJcxxglBmxxgl()
{
	CDlgDepartment m_hDlg;
	m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglYgxxgl()
{
```

```
        CDlgEmployee m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglXtgngl()
{
        CDlgSysFunctionInfo m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglJsgl()
{
        CDlgRoleinfo m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglCzygl()
{
        CDlgOperator m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglSpxxgl()
{
        CDlgSpxxgl m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnJcxxglKcxxgl()
{
        CDlgKcgl m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnCgglCgsq()
{
        CDlgStockApply m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnCgglCgrk()
{
        CDlgIntostorage m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnCgglRkth()
{
        CDlgCancelin m_hDlg;
        m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnCgcxCgsqcx()
{
        m_nCxSelected = 1;
        CDlgPrint m_hDlg;
        m_hDlg.DoModal();
```



```
}
void CMerchandiseStoreDlg::OnCgcxCgrkcx()
{
	m_nCxSelected = 2;
	CDlgPrint m_hDlg;
	m_hDlg.DoModal();
}
void CMerchandiseStoreDlg::OnCgcxRkthcx()
{
	m_nCxSelected = 3;
	CDlgPrint m_hDlg;
	m_hDlg.DoModal();
}
```

在 MerchandiseStoreDlg.h 头文件中加入如下代码：

```
class CMerchandiseStoreDlg : public CDialog
{
public:
	CMerchandiseStoreDlg(CWnd* pParent = NULL);
	enum { IDD = IDD_MERCHANDISESTORE_DIALOG };
protected:
	virtual void DoDataExchange(CDataExchange* pDX);
protected:
	HICON m_hIcon;

	//{{AFX_MSG(CMerchandiseStoreDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnJcxxglBmxxgl();
	afx_msg void OnJcxxglYgxxgl();
	afx_msg void OnJcxxglXtgngl();
	afx_msg void OnJcxxglJsgl();
	afx_msg void OnJcxxglCzygl();
	afx_msg void OnJcxxglSpxxgl();
	afx_msg void OnJcxxglKcxxgl();
	afx_msg void OnCgglCgsq();
	afx_msg void OnCgglCgrk();
	afx_msg void OnCgglRkth();
	afx_msg void OnCgcxCgsqcx();
	afx_msg void OnCgcxCgrkcx();
	afx_msg void OnCgcxRkthcx();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
#endif
```

5.5.3 菜单 实现 程

BCMenu 函数实现代码如下：

```
BCMenu::BCMenu()
{
    m_bDynIcons = FALSE;
    disable_old_style=FALSE;
    m_iconX = 16;
    m_iconY = 15;
    m_selectcheck = -1;
    m_unselectcheck = -1;
    checkmaps=NULL;
    checkmapsshare=FALSE;
    m_bitmapBackground=RGB(192,192,192);
    m_bitmapBackgroundFlag=FALSE;
    GetCPInfo(CP_ACP,&CPInfo);
    m_loadmenu=FALSE;
}


BCMenu::~BCMenu()
{
    DestroyMenu();
}

BOOL BCMenu::IsNewShell ()
{
    return (g_Shell>=Win95);
}

BOOL BCMenu::IsWinXPLuna()
{
    if(g_Shell==WinXP){
        if(IsWindowsClassicTheme())return(FALSE);
        else return(TRUE);
    }
    return(FALSE);
}

BOOL BCMenu::IsLunaMenuStyle()
{
    if(IsWinXPLuna()){
        if(xp_drawmode==BCMENU_DRAWMODE_XP)return(TRUE);
    }
    else{
        if(original_drawmode==BCMENU_DRAWMODE_XP)return(TRUE);
    }
```

```
	return(FALSE);
}

BCMenuData::~BCMenuData()
{
	if(bitmap)
		delete(bitmap);

	delete[] m_szMenuText;
}
```

功能实现代码如下：

```
void BCMenuData::SetWideString(const wchar_t *szWideString)
{
	delete[] m_szMenuText;

	if (szWideString)
	{
		m_szMenuText = new wchar_t[sizeof(wchar_t)*(wcslen(szWideString)+1)];
		if (m_szMenuText)
			wcscpy(m_szMenuText,szWideString);
	}
	else
		m_szMenuText=NULL;
}

BOOL BCMenu::IsMenu(CMenu *submenu)
{
	int m;
	int numSubMenus = m_AllSubMenus.GetUpperBound();
	for(m=0;m<=numSubMenus;++m){
		if(submenu->m_hMenu==m_AllSubMenus[m])return(TRUE);
	}
	return(FALSE);
}

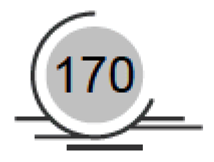
BOOL BCMenu::IsMenu(HMENU submenu)
{
	int m;
	int numSubMenus = m_AllSubMenus.GetUpperBound();
	for(m=0;m<=numSubMenus;++m){
		if(submenu==m_AllSubMenus[m])return(TRUE);
	}
	return(FALSE);
}

BOOL BCMenu::DestroyMenu()
{
```

```
	int m,n;
	int numAllSubMenus = m_AllSubMenus.GetUpperBound();
	for(n = numAllSubMenus; n>= 0; n--){
		if(m_AllSubMenus[n]==this->m_hMenu)m_AllSubMenus.RemoveAt(n);
	}
	int numSubMenus = m_SubMenus.GetUpperBound();
	for(m = numSubMenus; m >= 0; m--){
		numAllSubMenus = m_AllSubMenus.GetUpperBound();
		for(n = numAllSubMenus; n>= 0; n--){
			if(m_AllSubMenus[n]==m_SubMenus[m])m_AllSubMenus.RemoveAt(n);
		}
		CMenu *ptr=FromHandle(m_SubMenus[m]);
		BOOL flag=ptr->IsKindOf(RUNTIME_CLASS(BCMenu));
		if(flag)delete((BCMenu *)ptr);
	}
	m_SubMenus.RemoveAll();
	int numItems = m_MenuList.GetUpperBound();
	for(m = 0; m <= numItems; m++)delete(m_MenuList[m]);
	m_MenuList.RemoveAll();
	if(checkmaps&&!checkmapsshare){
		delete checkmaps;
		checkmaps=NULL;
	}
	return(CMenu::DestroyMenu());
};

int BCMenu::GetMenuDrawMode(void)
{
	if(IsWinXPLuna())return(xp_drawmode);
	return(original_drawmode);
}

BOOL BCMenu::GetSelectDisableMode(void)
{
	if(IsLunaMenuStyle())return(xp_select_disabled);
	return(original_select_disabled);
}
void CDlgIntostorage::OnBtnAdd()
{
	switch(m_hTabIntostorage.GetCurSel())
	{
	case 0:
		{
			break;
		}
	case 1:
		{
			TabCtrlOfSelect(0);
			break;
```





```
        }
    }
    m_hEditID.SetWindowText("");
    m_hEditOperatorID.SetWindowText("");
    m_hEditOrderform.SetWindowText("");

    m_hEditID.SetFocus();
    m_hBtnSave.EnableWindow();

}
```

5.6　采购管理模块及按键设计



采购管理模块主要包含商品采购申请、采购入库和入库退货功能，涉及商品采购信息的添加、修改、删除和保存等操作。下面以商品采购申请模块为例，介绍采购管理程序的设计方法。

5.6.1　购申请模块概

采购申请是采购部门根据计划部门的物料需求计划，向上级业务主管部门提交购货申请。采购申请模块运行效果如图 5.14 所示。

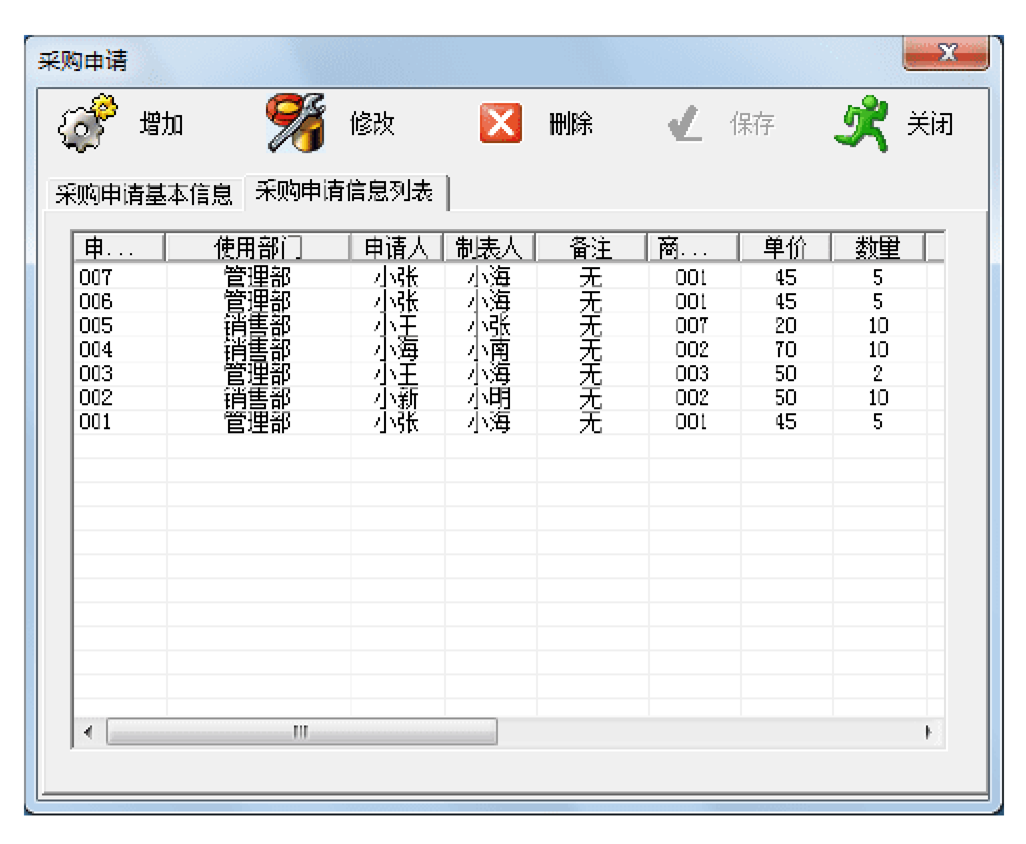


图 5.14　采购申请模块运行效果

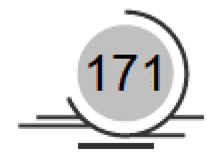
5.6.2　购申请模块技术分析

在实现商品采购管理模块时需要用到派生于 Cdialog 的 CDlgStockApply 类，派生于 Cdialog 的 CDlgStockApply 类同样可以重载 Cdialog 的虚函数 OnInitDialog，在窗口初始化时调用 InitCtrlData 方法加载数据，相关代码如下：

```
BOOL CDlgStockApply::OnInitDialog()
{
    CDialog::OnInitDialog();
    InitCtrlData();
    return TRUE;
}
```

5.6.3　购申请模块实现　程

（1）在工作区的类视图的 MerchandiseStore classes 节点上右击，在弹出的快捷菜单中选择 Insert



Dialog 命令，打开 Dialog 对话框，如图 5.15 所示。

（2）在 Dialog 对话框中右击，在弹出的快捷菜单中选择 Properties 命令，将弹出 Dialog Properties 对话框，如图 5.16 所示。

（3）选择 General 选项卡，设置窗体 ID 为 IDD_DLG_STOCKAPPLY，设置 Caption 为“采购申请”，如图 5.17 所示。

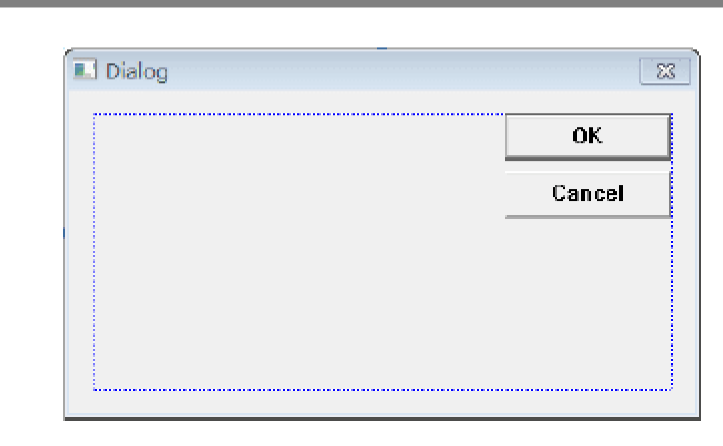


图 5.15　Dialog 对话框

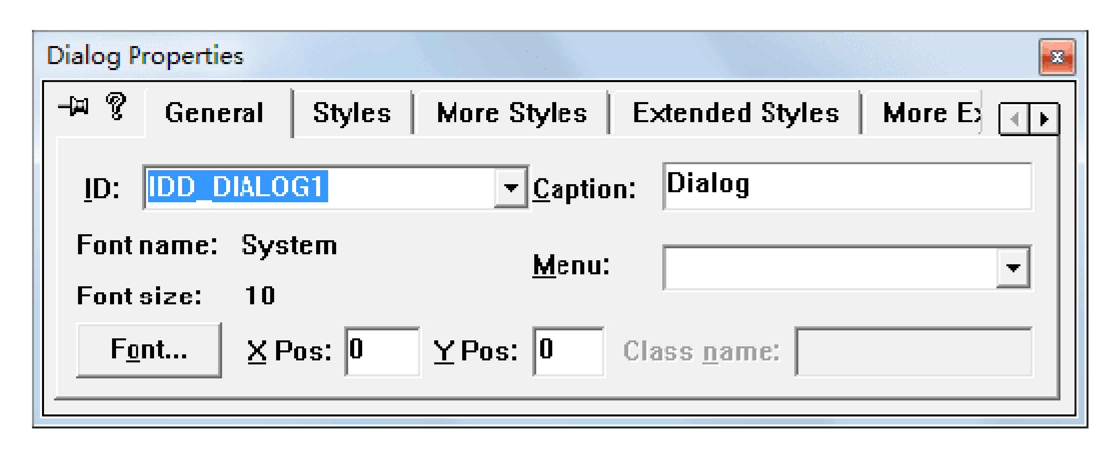




图 5.16　Dialog Properties 对话框



图 5.17　设置对话框的 ID 和标题

（4）向对话框中添加一个群组框资源、一个静态文本资源、一个编辑框资源、一个列表控件资源和一个时间控件，如图 5.18 所示。

（5）更改静态文本资源的标题属性为“申请单号”，如图 5.19 所示。

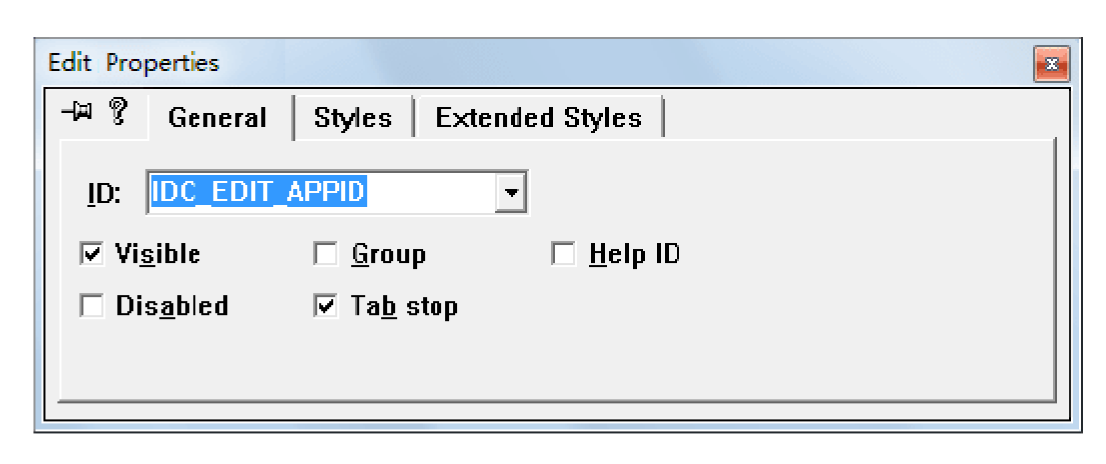




图 5.18　“采购申请”窗体设计结果　　　　图 5.19　更改静态文本资源显示标题

（6）更改编辑框资源的 ID 为 IDC_EDIT_APPID，如图 5.20 所示。

（7）更改群组框资源的 ID 为 IDC_TAB_StockApply，如图 5.21 所示。



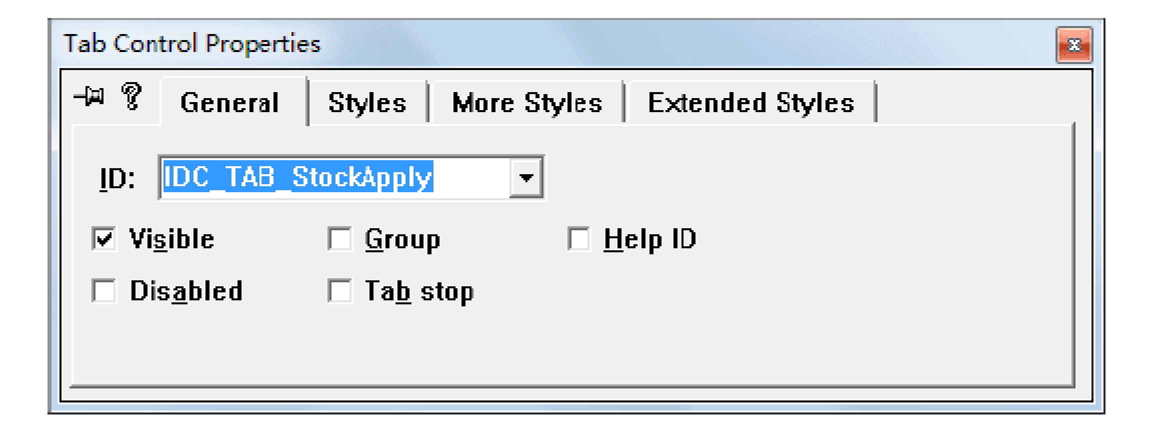
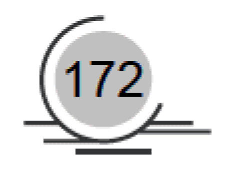


图 5.20　更改编辑框资源的 ID　　　　图 5.21　更改群组框资源的 ID





（8）更改列表控件资源的 ID 为 IDC_LIST_StockApply，如图 5.22 所示。

（9）以此类推，添加多个静态文本资源和编辑框资源，设置相关属性，完成资源创建，如图 5.23 所示。

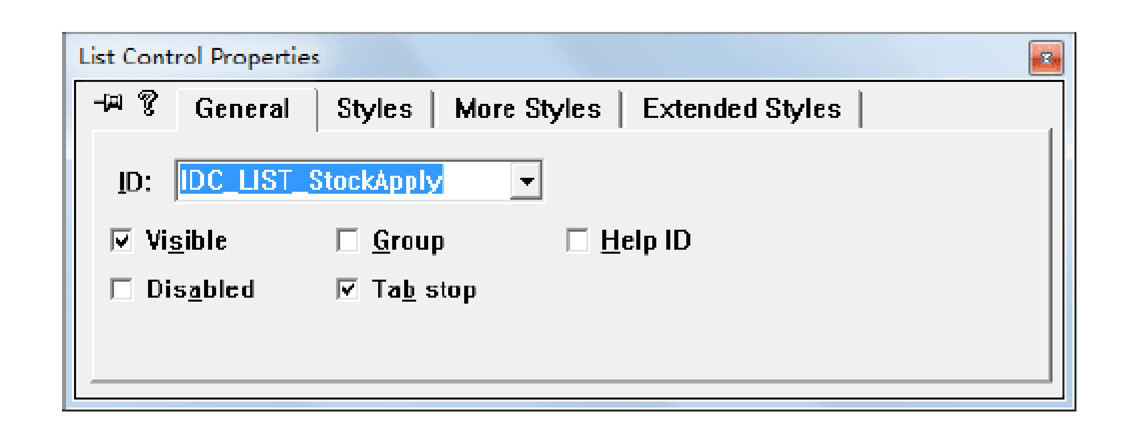


图 5.22　更改列表控件资源的 ID



图 5.23　静态文本资源和编辑框资源设计结果

（10）设置各主要资源属性，如表 5.7 所示。

表 5.7　资源属性设置

对 象 名 称	资 源 符 号	资 源 变	资 源 属 性
Static Box	IDC_STATIC		更改相应标题
Edit Box	IDC_EDIT_OPNAME	m_hEditName	其他以此类推
Button	IDC_BTN_ADD	m_hBtnAdd	标题：增加
Button	IDC_BTN_DEL	m_hBtnDel	标题：删除
Button	IDC_BTN_MOD	m_hBtnMod	标题：修改
Button	IDC_BTN_SAVE	m_hBtnSave	标题：保存
Button	IDOK	m_hBtnOk	标题：关闭
List ctrl	IDC_LIST_OP	m_hListOp	View：Report
Tab Ctrl	IDC_TAB_OP	m_hTabOp	默认

InitCtrlData 成员函数用于初始化所有控件内容及属性，代码如下：

```
void CDlgStockApply::InitCtrlData()
{

    m_hTabStockApply.InsertItem(0, "  购申请基本信息");
    m_hTabStockApply.InsertItem(1, "  购申请信息列表");
    m_hTabStockApply.ShowWindow(TRUE);

    m_hListStockApply.InsertColumn(0, "申请单号", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(1, "使用      ", LVCFMT_CENTER, 100);
    m_hListStockApply.InsertColumn(2, "申请人", LVCFMT_CENTER, 50);
```

```
    m_hListStockApply.InsertColumn(3, "制表人", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(4, "备注", LVCFMT_CENTER, 60);
    m_hListStockApply.InsertColumn(5, "商品编号", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(6, "单价", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(7, "数    ", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(8, "      ", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(9, "折扣", LVCFMT_CENTER, 50);
    m_hListStockApply.InsertColumn(10, "用    ", LVCFMT_CENTER, 100);
    m_hListStockApply.InsertColumn(11, "供应商", LVCFMT_CENTER, 100);
    m_hListStockApply.InsertColumn(12, "订单号", LVCFMT_CENTER, 100);
    m_hListStockApply.InsertColumn(13, "库存名称", LVCFMT_CENTER, 100);
    m_hListStockApply.SetExtendedStyle(m_hListStockApply.GetStyle() | LVS_EX_FULLROWSELECT);


    m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
    m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    m_hBtnSave.SetIcon(IDI_ICON_OK);
    m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
    m_hBtnSave.EnableWindow(FALSE);

    m_hBtnDel.SetIcon(IDI_ICON_DEL);
    m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

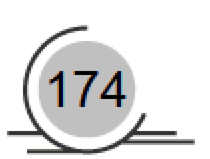
    m_hBtnAdd.SetIcon(IDI_ICON_ADD);
    m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    m_hBtnMod.SetIcon(IDI_ICON_MOD);
    m_hBtnMod.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnMod.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));


    TabCtrlOfSelect(1);
    m_hDatabase.InitEmployeeData(&m_hCmbDepartment);
}
```

TabCtrlOfSelect 成员函数用于切换选项卡，代码如下：

```
void CDlgStockApply::TabCtrlOfSelect(int m_nSelected)
{
    switch(m_nSelected)
    {
    case 0:
        {
            m_hTabStockApply.SetCurSel(0);
```



```
                m_hEditUnitprice.ShowWindow(TRUE);
                m_hEditSum.ShowWindow(TRUE);
                m_hEditStockname.ShowWindow(TRUE);
                m_hEditRebate.ShowWindow(TRUE);
                m_hEditPurpose.ShowWindow(TRUE);
                m_hEditProviderid.ShowWindow(TRUE);
                m_hEditProposer.ShowWindow(TRUE);
                m_hEditPaymoney.ShowWindow(TRUE);
                m_hEditOrderform.ShowWindow(TRUE);
                m_hEditMerchandiseID.ShowWindow(TRUE);
                m_hEditMemo.ShowWindow(TRUE);
                m_hEditBillmaker.ShowWindow(TRUE);
                m_hEditAppid.ShowWindow(TRUE);
                m_hCmbDepartment.ShowWindow(TRUE);
                m_hListStockApply.ShowWindow(FALSE);
                m_hEditAppid.SetFocus();
                break;
            }
        case 1:
            {
                m_hTabStockApply.SetCurSel(1);
                m_hEditUnitprice.ShowWindow(FALSE);
                m_hEditSum.ShowWindow(FALSE);
                m_hEditStockname.ShowWindow(FALSE);
                m_hEditRebate.ShowWindow(FALSE);
                m_hEditPurpose.ShowWindow(FALSE);
                m_hEditProviderid.ShowWindow(FALSE);
                m_hEditProposer.ShowWindow(FALSE);
                m_hEditPaymoney.ShowWindow(FALSE);
                m_hEditOrderform.ShowWindow(FALSE);
                m_hEditMerchandiseID.ShowWindow(FALSE);
                m_hEditMemo.ShowWindow(FALSE);
                m_hEditBillmaker.ShowWindow(FALSE);
                m_hEditAppid.ShowWindow(FALSE);
                m_hCmbDepartment.ShowWindow(FALSE);
                m_hListStockApply.ShowWindow(TRUE);
                break;
            }
        }
        m_hDatabase.ListStockApplyToCtrl(&m_hListStockApply);
}
```

处理“增加”按钮的单击事件，判断当前所处选项卡，切换至增加内容页面，清空所有内容，代码如下：

```
{
        switch(m_hTabCancelSell.GetCurSel())
        {
        case 0:
```

```
            {
                  break;
            }
      case 1:
            {
                  TabCtrlOfSelect(0);
                  break;
            }
      }
      m_hEditUnitPrice.SetWindowText("");
      m_hEditSumTotal.SetWindowText("");
      m_hEditStockName.SetWindowText("");
      m_hEditRebate.SetWindowText("");
      m_hEditPayMoney.SetWindowText("");
      m_hEditOperator.SetWindowText("");
      m_hEditMerchandiseID.SetWindowText("");
      m_hEditFactMoney.SetWindowText("");
      m_hEditCustomer.SetWindowText("");
      m_hEditCancelID.SetWindowText("");
      m_hEditNumbers.SetWindowText("");
      m_hEditCancelID.SetFocus();
      m_hBtnSave.EnableWindow();
}
```

处理“修改”按钮的命令消息，代码如下：

```
{
      switch(m_hTabCancelSell.GetCurSel())
      {
      case 0:
            {
                  break;
            }
      case 1:
            {
                  if(m_hListCancelSell.GetSelectionMark() == -1)
                  {//未被　中
                        MessageBox("请　择欲修改条目!");
                        return;
                  }
                  break;
            }
      }
      TabCtrlOfSelect(0);
      m_hBtnSave.EnableWindow();
      m_hEditCancelID.SetFocus();
}
```

处理“删除”按钮的命令消息，删除当前记录。其中，DeleteDataWhere 为数据库操作自定义函数，请查看 5.4.3 节数据库封装类实现过程。

```
{
	if(m_hListCancelSell.GetSelectionMark() == -1)
	{//未被　中
		MessageBox("请　择欲删　条目!");
		return;
	}
	char m_szCancelID[30+1];
	m_hListCancelSell.GetItemText(m_hListCancelSell.GetSelectionMark(), 0, m_szCancelID, sizeof(m_
szCancelID));
	m_hDatabase.DeleteDataWhere(SPXS, m_szCancelID);
	TabCtrlOfSelect(1);
}
```

处理“保存”按钮的命令消息，其中，UpdateOpData 为数据库操作自定义函数，主要代码如下：

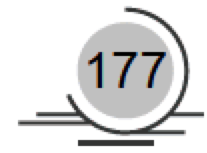
```
{
	char CancelID[30+1], Customer[30+1], ooperator[50+1], rebate[10+1], sumtotal[10+1], paymoney[10+1],
factmoney[10+1], intime[20+1]="", merchandiseID[30+1], unitPrice[10+1], numbers[10+1], stockname[30+1];
	m_hEditCancelID.GetWindowText(CancelID, sizeof(CancelID));
	m_hEditCustomer.GetWindowText(Customer, sizeof(Customer));
	m_hEditOperator.GetWindowText(ooperator, sizeof(ooperator));
	m_hEditRebate.GetWindowText(rebate, sizeof(rebate));
	m_hEditSumTotal.GetWindowText(sumtotal, sizeof(sumtotal));
	m_hEditPayMoney.GetWindowText(paymoney, sizeof(paymoney));
	m_hEditFactMoney.GetWindowText(factmoney, sizeof(factmoney));
	m_hEditMerchandiseID.GetWindowText(merchandiseID, sizeof(merchandiseID));
	m_hEditUnitPrice.GetWindowText(unitPrice, sizeof(unitPrice));
	m_hEditNumbers.GetWindowText(numbers, sizeof(numbers));
	m_hEditStockName.GetWindowText(stockname, sizeof(stockname));
	//保存修改，更新数据库
	m_hDatabase.UpdateSellData(CancelID,Customer,ooperator,rebate,sumtotal,paymoney,factmoney,intime,
merchandiseID,unitPrice,numbers,stockname);
	m_hBtnSave.EnableWindow(FALSE);
}
```

OnDblclkLISTStockApply 函数的代码如下：

```
void CDlgStockApply::OnDblclkLISTStockApply(NMHDR* pNMHDR, LRESULT* pResult)
{
	TabCtrlOfSelect(0);

	*pResult = 0;
}
```

OnClickListOp 为 ListCtrl 单击响应事件，其中，EditOpToCtrl 为数据库操作自定义函数，主要代



码如下：

```
{
	char CancelID[30+1];
	m_hListCancelSell.GetItemText(m_hListCancelSell.GetSelectionMark(), 0, CancelID, sizeof(CancelID)) ;
	m_hDatabase.EditSellToCtrl(CancelID, &m_hEditCancelID, &m_hEditUnitPrice, &m_hEditSumTotal,
&m_hEditStockName, &m_hEditRebate, &m_hEditPayMoney, &m_hEditOperator, &m_hEditNumbers,
&m_hEditMerchandiseID, &m_hEditFactMoney, &m_hEditCustomer) ;
	*pResult = 0;
}
//OnDblclkListOp 为 ListCtrl 双击响应事件，双击结果即为　　卡切换　程
{
	TabCtrlOfSelect(0);
	*pResult = 0;
}
```

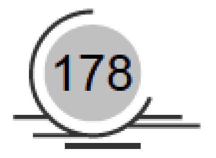
OnSelchangeTABStockApply 函数的代码如下：

```
void CDlgStockApply::OnSelchangeTABStockApply(NMHDR* pNMHDR, LRESULT* pResult)
{
	switch(m_hTabStockApply.GetCurSel())
	{
	case 0:
		{
			TabCtrlOfSelect(0);
			break;
		}
	case 1:
		{
			TabCtrlOfSelect(1);
			break;
		}
	}
	*pResult = 0;
}
```

对应的 DlgStockApply.h 头文件的代码如下：

```
class CDlgStockApply : public CDialog
{
public:
	CDlgStockApply(CWnd* pParent = NULL);

	//{{AFX_DATA(CDlgStockApply)
	enum { IDD = IDD_DLG_STOCKAPPLY };
	CTabCtrl m_hTabStockApply;
	CListCtrl m_hListStockApply;
	CButtonST m_hBtnOk;
	CEdit		m_hEditUnitprice;
	CEdit		m_hEditSum;
```



```
	CEdit		m_hEditStockname;
	CEdit		m_hEditRebate;
	CEdit		m_hEditPurpose;
	CEdit		m_hEditProviderid;
	CEdit		m_hEditProposer;
	CEdit		m_hEditPaymoney;
	CEdit		m_hEditOrderform;
	CEdit		m_hEditMerchandiseID;
	CEdit		m_hEditMemo;
	CEdit		m_hEditBillmaker;
	CEdit		m_hEditAppid;
	CDateTimeCtrl m_hDtpAppdate;
	CDateTimeCtrl m_hDtpBesttime;
	CComboBox	m_hCmbDepartment;
	CButtonST	m_hBtnSave;
	CButtonST	m_hBtnMod;
	CButtonST	m_hBtnDel;
	CButtonST	m_hBtnAdd;
	//}}AFX_DATA


	//{{AFX_VIRTUAL(CDlgStockApply)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);
	//}}AFX_VIRTUAL

protected:

	//{{AFX_MSG(CDlgStockApply)
	virtual BOOL OnInitDialog();
	afx_msg void OnBtnAdd();
	afx_msg void OnBtnDel();
	afx_msg void OnBtnMod();
	afx_msg void OnBtnSave();
	afx_msg void OnClickLISTStockApply(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnDblclkLISTStockApply(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnSelchangeTABStockApply(NMHDR* pNMHDR, LRESULT* pResult);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	void TabCtrlOfSelect(int m_nSelected);
	void InitCtrlData();
};

//{{AFX_INSERT_LOCATION}}

#endif
```

5.6.4　购物品操作模块实现　程

显示采购信息的代码如下：

```
void CDlgOperator::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

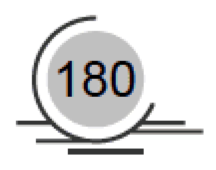
	m_hTabOperator.InsertItem(0, "操作员基本信息");
	m_hTabOperator.InsertItem(1, "操作员信息列表");
	m_hTabOperator.ShowWindow(TRUE);

	m_hListOperator.InsertColumn(0, "编号", LVCFMT_CENTER, 50);
	m_hListOperator.InsertColumn(1, "姓名", LVCFMT_CENTER, 80);
	m_hListOperator.InsertColumn(2, "密码", LVCFMT_CENTER, 80);
	m_hListOperator.InsertColumn(3, "角色", LVCFMT_CENTER, 80);;
	m_hListOperator.SetExtendedStyle(m_hListOperator.GetStyle() | LVS_EX_FULLROWSELECT);
	TabCtrlOfSelect(1);
	m_hDatabase.InitOperatorData(&m_hCmbRole);
}
```

5.6.5　购添加物品模块实现　程

映射代码如下：

```
BEGIN_MESSAGE_MAP(CDlgIntostorage, CDialog)
	//{{AFX_MSG_MAP(CDlgIntostorage)
	ON_NOTIFY(NM_CLICK, IDC_LIST_Intostorage, OnClickLISTIntostorage)
	ON_NOTIFY(NM_DBLCLK, IDC_LIST_Intostorage, OnDblclkLISTIntostorage)
	ON_NOTIFY(TCN_SELCHANGE, IDC_TAB_Intostorage, OnSelchangeTABIntostorage)
```



```
	ON_BN_CLICKED(IDD_BTN_ADD, OnBtnAdd)
	ON_BN_CLICKED(IDD_BTN_DEL, OnBtnDel)
	ON_BN_CLICKED(IDD_BTN_MOD, OnBtnMod)
	ON_BN_CLICKED(IDD_BTN_SAVE, OnBtnSave)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

利用 InitCtrlData 函数添加数据，实现代码如下：

```
void CDlgIntostorage::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnMod.SetIcon(IDI_ICON_MOD);
	m_hBtnMod.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnMod.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hTabIntostorage.InsertItem(0, "入库基本信息");
	m_hTabIntostorage.InsertItem(1, "入库信息列表");
	m_hTabIntostorage.ShowWindow(TRUE);

	m_hListIntostorage.InsertColumn(0, "入库单号", LVCFMT_CENTER, 100);
	m_hListIntostorage.InsertColumn(1, "订单号", LVCFMT_CENTER, 100);
	m_hListIntostorage.InsertColumn(2, "操作员", LVCFMT_CENTER, 100);
	m_hListIntostorage.InsertColumn(3, "入库时   ", LVCFMT_CENTER, 100);
	m_hListIntostorage.SetExtendedStyle(m_hListIntostorage.GetStyle() | LVS_EX_FULLROWSELECT);
	TabCtrlOfSelect(1);
}
```

选择删除条目信息代码如下：

```
void CDlgOperator::OnBtnDel()
{
```

```
	if(m_hListOperator.GetSelectionMark() == -1)
	{//未被  中
		MessageBox("请  择欲删  条目!");
		return;
	}
	char m_szID[30+1];
	m_hListOperator.GetItemText(m_hListOperator.GetSelectionMark(), 0, m_szID, sizeof(m_szID));
	m_hDatabase.DeleteDataWhere(OP, m_szID);
	m_hDatabase.ListOperatorToCtrl(&m_hListOperator);
}
```

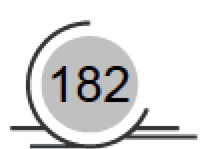
修改条目代码如下：

```
void CDlgOperator::OnBtnMod()
{
	switch(m_hTabOperator.GetCurSel())
	{
	case 0:
		{
			break;
		}
	case 1:
		{
			if(m_hListOperator.GetSelectionMark() == -1)
			{//未被  中
				MessageBox("请  择欲修改条目!");
				return;
			}
			break;
		}
	}
	TabCtrlOfSelect(0);
	m_hBtnSave.EnableWindow();
	m_hEditID.SetFocus();
}
```

保存修改，更新数据的代码如下：

```
void CDlgOperator::OnBtnSave()
{
	char m_szID[30+1], m_szName[30+1], m_szPassword[30+1], m_szRole[30+1];

	m_hEditID.GetWindowText(m_szID, sizeof(m_szID));
	m_hEditName.GetWindowText(m_szName, sizeof(m_szName));
	m_hEditPassword.GetWindowText(m_szPassword, sizeof(m_szPassword));
	m_hCmbRole.GetWindowText(m_szRole, sizeof(m_szRole));
```



```
        //保存修改，更新数据库
        m_hDatabase.UpdateOperatorData(m_szID,m_szName,m_szPassword,m_szRole);

        m_hBtnSave.EnableWindow(FALSE);
}
BOOL CDlgOperator::PreTranslateMessage(MSG* pMsg)
{

        return CDialog::PreTranslateMessage(pMsg);
}

void CDlgOperator::OnBtnAdd111()
{
        switch(m_hTabOperator.GetCurSel())
        {
        case 0:
                {
                        break;
                }
        case 1:
                {
                        TabCtrlOfSelect(0);
                        break;
                }
        }
        m_hEditID.SetWindowText("");
        m_hEditName.SetWindowText("");
        m_hEditPassword.SetWindowText("");
        m_hCmbRole.SetWindowText("");

        m_hBtnSave.EnableWindow();
}
```

5.6.6 按　设计

（1）首页 OnTop 函数实现代码如下：

```
void CPreParent::OnTop()
{
        m_PosPage = 1;
        pPreView->SetCurrentPage(m_nCountPage, m_PosPage);

        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, FALSE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, FALSE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, TRUE);
```

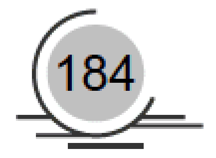
```
    UpdatePreViewWnd();
}
```

（2）上一页 OnPrevious 函数实现代码如下：

```
void CPreParent::OnPrevious()
{
    m_PosPage--;
    pPreView->SetCurrentPage(m_nCountPage, m_PosPage);
    if(m_PosPage<=1)
    {
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, FALSE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, FALSE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, TRUE);
    }
    else
    {
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, TRUE);
    }
    UpdatePreViewWnd();
}
```

（3）转到 OnGoto 函数实现代码如下：

```
void CPreParent::OnGoto()
{
    int nPage = 1;
    int m = m_nCount-m_OneCount;
    int n = m/m_NextCount;
    nPage += n;
    n = m%m_NextCount;
    if(n>0)
        nPage++;
    CPreGoto cpg;
    cpg.nMax = nPage;
    cpg.nCurPage = m_PosPage;
    if(cpg.DoModal())
    {
        m_PosPage = cpg.nGoto;
        pPreView->SetCurrentPage(m_nCountPage, m_PosPage);
        if(m_PosPage > 1 && m_PosPage< m_nCountPage)
        {
            m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, TRUE);
            m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, TRUE);
```



```
				m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, TRUE);
				m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, TRUE);
			}
			if(m_PosPage == 1)
				OnTop();
			if(m_PosPage == m_nCountPage)
				OnLast();
		}

		UpdatePreViewWnd();
}
```

（4）下一页 OnNext 函数实现代码如下：

```
void CPreParent::OnNext()
{
	m_PosPage++;
	pPreView->SetCurrentPage(m_nCountPage, m_PosPage);

	int nSpare = 0;
	nSpare = m_nCount - m_PosPage*m_NextCount;
	if(m_PosPage <= 2)
		nSpare +=(m_NextCount - m_OneCount);

	m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, TRUE);
	m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, TRUE);

	if(nSpare>0)
	{
		m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, TRUE);
		m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, TRUE);
	}
	else
	{
		m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, FALSE);
		m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, FALSE);
	}
	UpdatePreViewWnd();
}
```

（5）尾页 OnLast 函数实现代码如下：

```
void CPreParent::OnLast()
{
	m_PosPage = m_nCountPage;
	pPreView->SetCurrentPage(m_nCountPage, m_PosPage);

	m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_TOP, TRUE);
```

```
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_PREVIOUS, TRUE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_NEXT, FALSE);
        m_wndtoolbar.SendMessage(TB_ENABLEBUTTON, TBTN_LAST, FALSE);
        UpdatePreViewWnd();
}
```

（6）退出 OnExit 和打印 OnPrint 函数实现代码如下：

```
void CPreParent::OnExit()
{
        SendMessage(WM_SYSCOMMAND, SC_CLOSE, NULL);
}

void CPreParent::OnPrint()
{
        pPreView->PrintDoc();
}

BOOL CPreParent::DestroyWindow()
{
        if(IsWindow(m_wndtoolbar.m_hWnd))
                m_wndtoolbar.DestroyWindow();
        if(pPreView != NULL)
        {
                pPreView->DestroyWindow();
                delete pPreView;
        }

        return CDialog::DestroyWindow();
}

void CPreParent::UpdatePreViewWnd()
{
        pPreView->SendMessage(WM_PAINT, NULL, NULL);
}
```


视频讲解

5.7　基本信息模块设计

5.7.1　基本信息模块概

基本信息模块包括部门信息管理、员工信息管理、系统功能管理、角色管理、操作员管理、商品信息管理和库存信息管理。如图 5.24 所示是基本信息模块的主窗体。

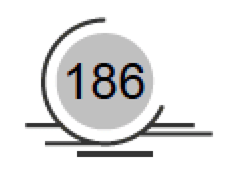






图 5.24　基本信息模块主窗体

5.7.2　基本信息模块技术分析

在实现商品采购管理模块时需要用到派生于 Cdialog 的 CDlgEmployee 类，派生于 Cdialog 的 CDlgEmployee 类同样可以重载 Cdialog 的虚函数 OnInitDialog，在窗口初始化时调用 InitCtrlData 方法加载数据，相关代码如下：

```
BOOL CDlgEmployee::OnInitDialog()
{
	CDialog::OnInitDialog();

	InitCtrlData();

	return TRUE;
}
```

5.7.3　基本信息模块实现　程

（1）部门管理信息的实现代码如下：

```
void CDlgDepartment::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
```

```
        m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
        m_hBtnSave.EnableWindow(FALSE);

        m_hBtnDel.SetIcon(IDI_ICON_DEL);
        m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hBtnAdd.SetIcon(IDI_ICON_ADD);
        m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hListDepartment.InsertColumn(0, "      名称", LVCFMT_CENTER, 400);
        m_hListDepartment.SetExtendedStyle(m_hListDepartment.GetStyle() | LVS_EX_FULLROWSELECT);

        m_hDatabase.ListDepartmentToCtrl(&m_hListDepartment);
}
```

（2）员工信息管理的实现代码如下：

```
void CDlgEmployee::InitCtrlData()
{
        m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
        m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hBtnSave.SetIcon(IDI_ICON_OK);
        m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
        m_hBtnSave.EnableWindow(FALSE);

        m_hBtnDel.SetIcon(IDI_ICON_DEL);
        m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hBtnAdd.SetIcon(IDI_ICON_ADD);
        m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hBtnMod.SetIcon(IDI_ICON_MOD);
        m_hBtnMod.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
        m_hBtnMod.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

        m_hTabEmployee.InsertItem(0, "员工基本信息");
        m_hTabEmployee.InsertItem(1, "员工信息列表");
        m_hTabEmployee.ShowWindow(TRUE);

        m_hListEmployee.InsertColumn(0, "员工编号", LVCFMT_CENTER, 80);
        m_hListEmployee.InsertColumn(1, "员工姓名", LVCFMT_CENTER, 50);
        m_hListEmployee.InsertColumn(2, "性别", LVCFMT_CENTER, 50);
```

```
	m_hListEmployee.InsertColumn(3, "年    ", LVCFMT_CENTER, 60);
	m_hListEmployee.InsertColumn(4, "学历", LVCFMT_CENTER, 100);
	m_hListEmployee.InsertColumn(5, "民族", LVCFMT_CENTER, 120);
	m_hListEmployee.InsertColumn(6, "毕业学    ", LVCFMT_CENTER, 100);
	m_hListEmployee.InsertColumn(7, "联系方式", LVCFMT_CENTER, 100);
	m_hListEmployee.InsertColumn(8, "家庭住址", LVCFMT_CENTER, 100);
	m_hListEmployee.InsertColumn(9, "        ", LVCFMT_CENTER, 100);
	m_hListEmployee.SetExtendedStyle(m_hListEmployee.GetStyle() | LVS_EX_FULLROWSELECT);
	TabCtrlOfSelect(1);
	m_hDatabase.InitEmployeeData(&m_hCmbDepartment);
}
```

（3）系统功能管理的实现代码如下：

```
void CDlgSysFunctionInfo::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hListSysFunctionInfo.InsertColumn(0, "系统功能", LVCFMT_CENTER, 400);
	m_hListSysFunctionInfo.SetExtendedStyle(m_hListSysFunctionInfo.GetStyle() | LVS_EX_FULLROWSELECT);

	m_hDatabase.ListSysFunctionInfoToCtrl(&m_hListSysFunctionInfo);
}
```

（4）角色管理的实现代码如下：

```
void CDlgRoleinfo::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
```

```
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hListRoleinfo.InsertColumn(0, "角色信息", LVCFMT_CENTER, 400);
	m_hListRoleinfo.SetExtendedStyle(m_hListRoleinfo.GetStyle() | LVS_EX_FULLROWSELECT);

	m_hDatabase.ListRoleInfoToCtrl(&m_hListRoleinfo);

}
```

（5）操作员管理的实现代码如下：

```
void CDlgOperator::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

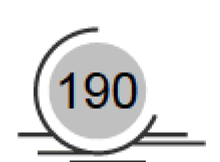
	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hTabOperator.InsertItem(0, "操作员基本信息");
	m_hTabOperator.InsertItem(1, "操作员信息列表");
	m_hTabOperator.ShowWindow(TRUE);

	m_hListOperator.InsertColumn(0, "编号", LVCFMT_CENTER, 50);
	m_hListOperator.InsertColumn(1, "姓名", LVCFMT_CENTER, 80);
	m_hListOperator.InsertColumn(2, "密码", LVCFMT_CENTER, 80);
	m_hListOperator.InsertColumn(3, "角色", LVCFMT_CENTER, 80);
```



```
    m_hListOperator.SetExtendedStyle(m_hListOperator.GetStyle() | LVS_EX_FULLROWSELECT);
    TabCtrlOfSelect(1);
    m_hDatabase.InitOperatorData(&m_hCmbRole);
}
```

（6）商品信息管理实现代码如下：

```
void CDlgSpxxgl::InitCtrlData()
{
    m_hTabSpxx.InsertItem(0, "商品基本信息");
    m_hTabSpxx.InsertItem(1, "商品信息列表");
    m_hTabSpxx.ShowWindow(TRUE);

    m_hListSpxx.InsertColumn(0, "ID", LVCFMT_CENTER, 30);
    m_hListSpxx.InsertColumn(1, "商品名称", LVCFMT_CENTER, 100);
    m_hListSpxx.InsertColumn(2, "规格", LVCFMT_CENTER, 50);
    m_hListSpxx.InsertColumn(3, "助记码", LVCFMT_CENTER, 50);
    m_hListSpxx.InsertColumn(4, "   认规格", LVCFMT_CENTER, 60);
    m_hListSpxx.InsertColumn(5, "厂家", LVCFMT_CENTER, 120);
    m_hListSpxx.InsertColumn(6, "备注", LVCFMT_CENTER, 100);
    m_hListSpxx.SetExtendedStyle(m_hListSpxx.GetStyle() | LVS_EX_FULLROWSELECT);

    m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
    m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    m_hBtnSave.SetIcon(IDI_ICON_OK);
    m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
    m_hBtnSave.EnableWindow(FALSE);

    m_hBtnDel.SetIcon(IDI_ICON_DEL);
    m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    m_hBtnAdd.SetIcon(IDI_ICON_ADD);
    m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    m_hBtnMod.SetIcon(IDI_ICON_MOD);
    m_hBtnMod.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
    m_hBtnMod.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

    TabCtrlOfSelect(1);
}
```

（7）库存信息管理的实现代码如下：

```
void CDlgKcgl::InitCtrlData()
{
	m_hBtnOk.SetIcon(IDI_ICON_CLOSE);
	m_hBtnOk.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnOk.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnSave.SetIcon(IDI_ICON_OK);
	m_hBtnSave.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnSave.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));
	m_hBtnSave.EnableWindow(FALSE);

	m_hBtnDel.SetIcon(IDI_ICON_DEL);
	m_hBtnDel.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnDel.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnAdd.SetIcon(IDI_ICON_ADD);
	m_hBtnAdd.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnAdd.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hBtnMod.SetIcon(IDI_ICON_MOD);
	m_hBtnMod.OffsetColor(CButtonST::BTNST_COLOR_BK_IN, shBtnColor);
	m_hBtnMod.SetColor(CButtonST::BTNST_COLOR_FG_IN, RGB(0, 128, 0));

	m_hTabKc.InsertItem(0, "库存基本信息");
	m_hTabKc.InsertItem(1, "库存信息列表");
	m_hTabKc.ShowWindow(TRUE);

	m_hListKc.InsertColumn(0, "库存编号", LVCFMT_CENTER, 100);
	m_hListKc.InsertColumn(1, "库存名称", LVCFMT_CENTER, 100);
	m_hListKc.InsertColumn(2, "库存数    ", LVCFMT_CENTER, 100);
	m_hListKc.SetExtendedStyle(m_hListKc.GetStyle() | LVS_EX_FULLROWSELECT);
	TabCtrlOfSelect(1);
}
```

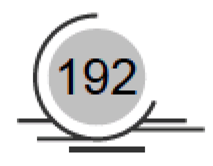

视频讲解

5.8 实现系统及单元测试

5.8.1 实现完整系统

在 MerchandiseStore.cpp 文件中填写以下代码，用来实现完整系统：

```
BEGIN_MESSAGE_MAP(CMerchandiseStoreApp, CWinApp)
	//{{AFX_MSG_MAP(CMerchandiseStoreApp)
	//}}AFX_MSG
```



```
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

CMerchandiseStoreApp::CMerchandiseStoreApp()
{
}

CMerchandiseStoreApp theApp;
CDatabase m_hDatabase;

BOOL CMerchandiseStoreApp::InitInstance()
{
	AfxEnableControlContainer();
	CoInitialize(NULL);

#ifdef _AFXDLL
	Enable3dControls();
#else
	Enable3dControlsStatic();
#endif

	if(!m_hDatabase.InitData())
	{
		MessageBox(NULL, "数据库访  失败,程序异常关  !", "出  啦", MB_OK);
		exit(1);
	}

	CMerchandiseStoreDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
	}
	else if (nResponse == IDCANCEL)
	{
	}

	return FALSE;
}
```

5.8.2　单元测试

1．怎样取得当前日期

在 MFC 当中，提供了日期类 CTime，这个类中有一个成员函数 GetCurrentTime 可以获得当前系统时间，代码如下：

```
CTime time;
time=time.GetCurrentTime();
```

time 是当前时间。

2．怎样取得当前　径

取得当前路径也是非常轻松的事情，可以通过 API 函数 GetCurrentDirectory 取得，代码如下：

```
char str[_MAX_DIR];
::GetCurrentDirectory(sizeof(str),str);
```

3．常见　误

从某种意义上来说，调试程序比编写程序更难。对某些函数、命令理解不深，在编写程序中疏忽大意等都会导致程序调试中出现问题。编程中出现的问题有三大类：语法错误、逻辑错误和例外错误。

语法错误是其中最容易出现和纠正的，编译代码是查找大多数错误最快的方法。强大的 Visual C++编辑器会将多数错误定位到出错的行。

逻辑错误解决起来麻烦一些，编译程序时并不会发现，有时严重的逻辑错误可能会停止程序的执行。

例外错误是由程序直接控制外部环境引起的。例如，程序因为找不到它所需要的文件而运行失败，也许这个文件已被删除或移动。

4．截获回　后的潜在

在前面已经介绍过，为了保证用户在某控件上按下 Enter 键时，都会将焦点定位到下一个控件上，可以通过将 Enter 键转换成 Tab 键来实现。虽然目的达到了，却产生了一个负面问题，如果需要再次截获 Enter 键，就麻烦了。下面举例说明。

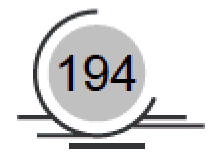
以下代码是在 CBaseComboBox 中截获 Enter 键，并将其更改为 Tab 键。

```
BOOL CBaseComboBox::PreTranslateMessage(MSG* pMsg)
{
	if(pMsg->message==WM_KEYDOWN && pMsg->wParam==13)
		pMsg->wParam=9;
	return CComboBox::PreTranslateMessage(pMsg);
}
```

在对话框中，需要截获 CBaseComboBox 类对象的 Enter 键响应。

```
if(pMsg->message==WM_KEYDOWN&& pMsg->wParam==13)
{
	……//使表格第一个单元格获得焦点
}
```

在表格中按 Enter 键，焦点却不移动。采用跟踪调试，如图 5.25 所示。



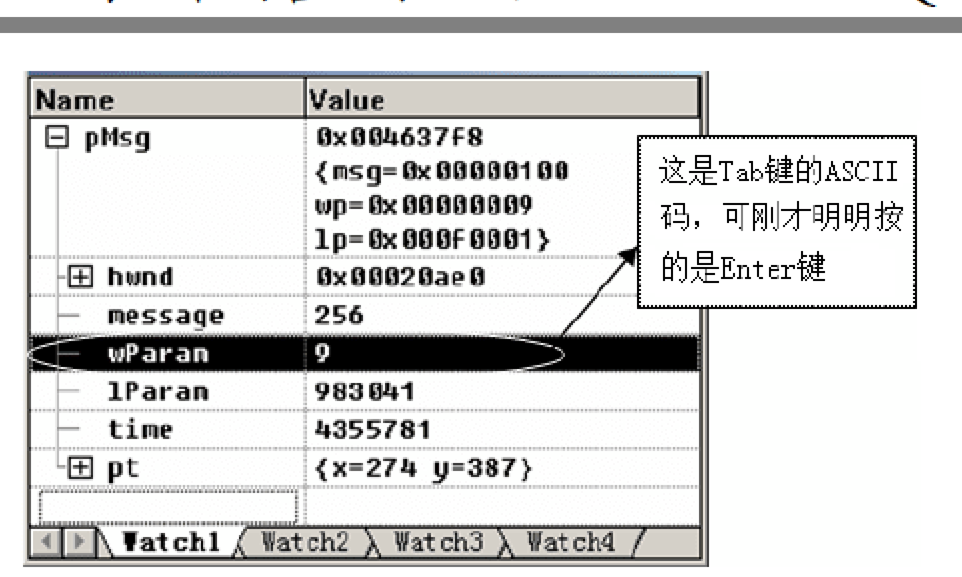


图 5.25　Watch 对话框

实际上在按下 Enter 键时，CBaseComboBox 类就已经截获了 Enter 键信息，并将其更改为 Tab 键，在截获 CBaseComboBox 类对象的键盘消息时又怎么会有 Enter 键信息呢？修改后代码如下：

```
if(pMsg->message==WM_KEYDOWN&& pMsg->wParam== VK_TAB)
{
    …//使表格第一个单元格获得焦点
}
```

5.9　项目文件清单

商品采购管理系统客户端项目文件清单如表 5.8 所示。

表 5.8　客户端　目文件清单

文件名称	文件类型	文件描	文件名称	文件类型	文件描
MerchandiseStore.dsp	工程文件	客户端工程文件	DlgOperator.cpp	源文件	操作员信息管理
MerchandiseStore.dsw	工作区文件	客户端工作区文件	DlgPrint.cpp	源文件	打印
MerchandiseStore.rc	资源文件	客户端资源文件	DlgRoleinfo.cpp	源文件	角色信息管理
MerchandiseStoreDlg.h	源文件	商品采购管理系统	DlgSpxxgl.cpp	源文件	商品信息管理
DlgCancelin.cpp	源文件	入库退货管理	DlgStockApply.cpp	源文件	采购申请
DlgDepartment.cpp	源文件	部门信息管理	DlgSysFunctionInfo.cpp	源文件	系统功能管理
DlgEmployee.cpp	源文件	员工信息管理	PreGoto.cpp	源文件	转到页
DlgIntostorage.cpp	源文件	采购入库管理	PreParent.cpp	源文件	打印预览
DlgKcgl.cpp	源文件	库存信息管理			

5.10　本章总结

本章通过一个完整的商品采购管理系统详细讲解了一个系统的开发流程。通过本章的学习，读者应该掌握 SQL Server2014 的使用方法和对数据库封装类等知识，希望对读者日后的程序开发有所帮助。

第 6 章

文档管理系统

（Visual Studio 2017+SQL Server 2014 实现）

文档管理即文件的制作、修改、传递、签订、保存、销毁和存档等一系列操作。文档管理系统是企业经营管理中不可缺少的部分，通过文档信息管理系统，可以实现文档自动化管理的目标，为企业提供了安全、可靠、开放和高效的文档管理功能，不仅方便了日常操作，而且避免了手工管理中一系列错误的发生，提高了企业的办公效率和企业文件管理的综合水平。

通过学习本章，读者可以学到：

- **利用 ADO 对象连接数据库**
- **利用 CFileStatus 类获得文档属性**
- **利用 Word 类操作 Word 文档**
- **备份和还原数据库**



配置说明



6.1　开 发 背 景



目前，大多数文档管理系统在实现了企业各部门日常文件管理的基本功能之外，还增设了很多新功能用以满足文档管理电子化、标准化的新要求。功能强大的档案查询模块大大方便了管理者日常查找文档的工作，解决了传统管理中查找困难、查找耗时等问题。使用现代化的文档管理系统满足了企业“无纸化”办公的要求，实现了通过计算机对文档管理全程跟踪的目标。

6.2　需 求 分 析

根据市场的需求，要求系统具有以下功能。

☑　处理大量的复合文档型的数据信息。
☑　通过系统查看文档内容和属性。
☑　通过系统可以完成对文档的一系列日常操作。
☑　保证系统的安全性和可靠性。
☑　由于操作人员的计算机操作能力普遍较差，因此要求系统具有良好的人机交互界面。
☑　完全人性化设计，无须专业人士指导，即可操作本系统。
☑　系统具有数据备份及数据还原功能，能够保证系统数据的安全性。

6.3　系 统 设 计

6.3.1　系统目标

本系统是根据中小企业的实际需求开发的，完全能够实现企业对制度文档的自动化管理，通过本系统可以达到以下目标。

☑　系统运行稳定，安全可靠。
☑　界面设计美观，人机交互界面友好。
☑　信息查询灵活、方便、快捷、准确，数据存储安全可靠。
☑　操作员可以随时修改自己的口令。
☑　对用户输入的数据，系统进行严格的数据检验，尽可能排除人为的错误。
☑　数据保密性强，为每个用户设置相应的权限级别。

6.3.2　系统功能结构

文档管理系统的功能结构如图 6.1 所示。

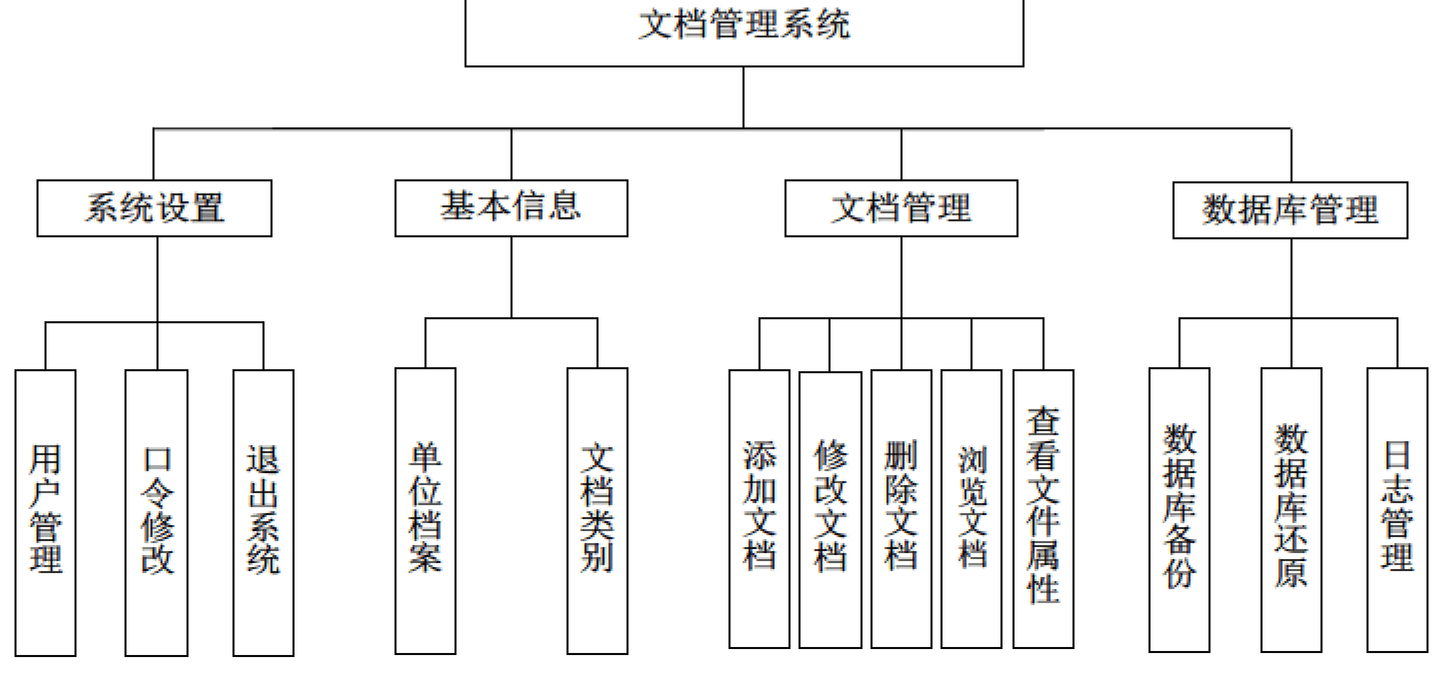


图 6.1　文档管理系统功能结构图

6.3.3　系统预览

文档管理系统的主界面如图 6.2 所示。

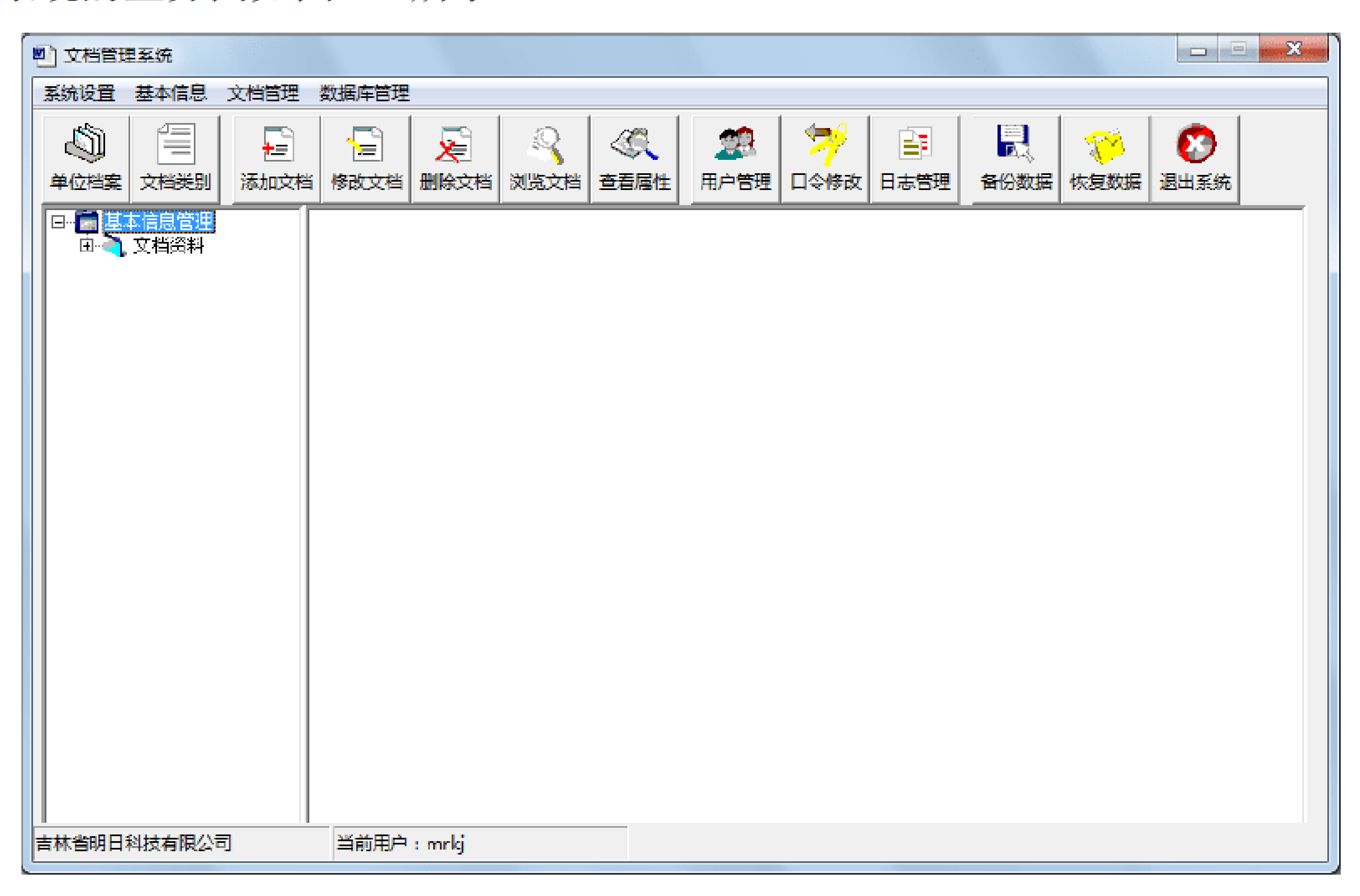


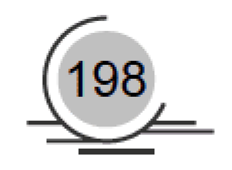
图 6.2　文档管理系统的主界面

6.3.4　业务流程图

文档管理系统的业务流程图如图 6.3 所示。

6.3.5　数据库设计

本系统的数据库采用 SQL Server 2014，系统数据库的名称为 WenDGL。数据库中包含 5 张数据表。下面分别给出数据表概要说明和主要数据表的结构。





1．数据表概要说明

为使读者对本系统后台数据库中的数据表有一个更清晰的认识，在此特别设计了一个数据表树形结构图，该数据表树形结构图中包含了系统中所有的数据表，如图 6.4 所示。

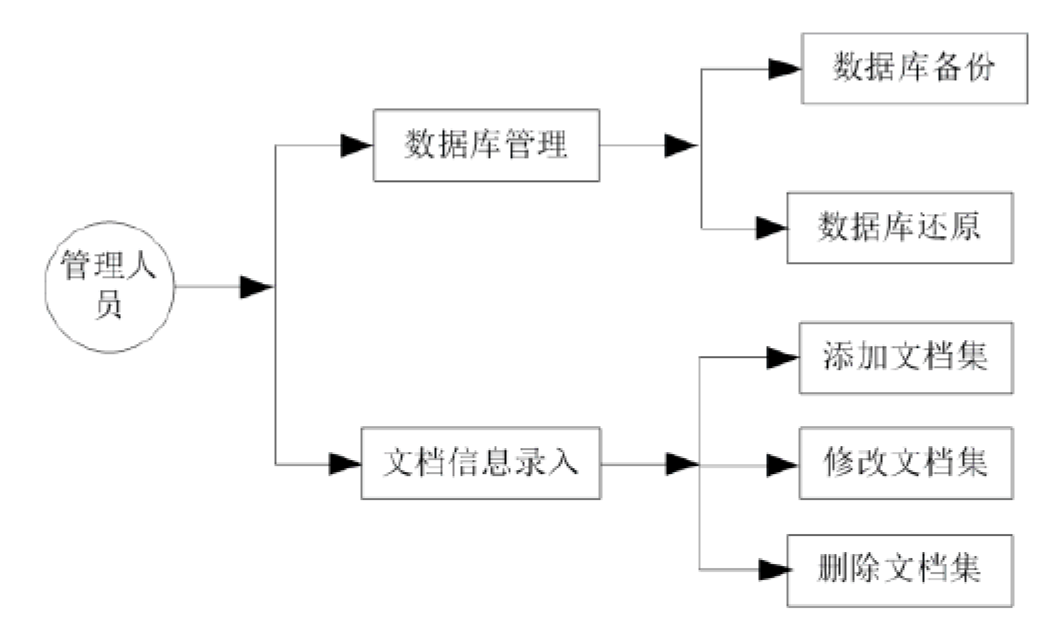


图 6.3　文档管理系统业务流程图

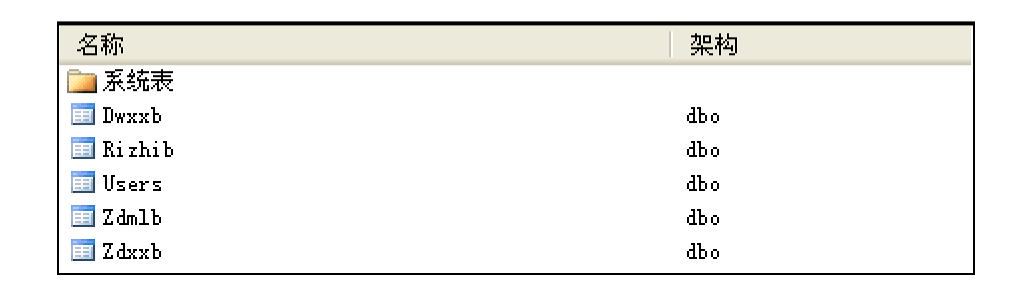


图 6.4　数据表树形结构图

2．主要数据表的结构

在此给出数据表的表结构。

☑　单位表（Dwxxb）：用来存储企业信息。该表的结构如表 6.1 所示。

表 6.1　单位表

字　段　名	数 据 类 型	度	描
DWbh	int	4	单位编号
DWmc	varchar	50	单位名称
Lxr	varchar	50	联系人
Lxdh	varchar	50	联系电话
Lxdz	varchar	50	联系地址
Memo	varchar	200	备注

☑　类别表（Zdmlb）：用来存储在企业中创建的文档类别。该表的结构如表 6.2 所示。

表 6.2　类别表

字　段　名	数 据 类 型	度	描
DWbh	int	4	单位编号
LBbh	int	4	类别编号
LBmc	varchar	50	类别名称

☑　文档表（Zdxxb）：用来存储日常使用的文档信息。该表的结构如表 6.3 所示。

表 6.3　文档表

字　段　名	数 据 类 型	度	描
DWbh	int	4	单位编号
LBbh	int	4	类别编号

续表

字　段　名	数 据 类 型	度	描
WDbh	int	4	文档编号
WDmc	varchar	300	文档名称
GJz	varchar	200	关键字
WJlj	varchar	300	文件路径
Memo	varchar	200	备注
Tjrxm	varchar	50	添加人姓名

☑ 日志表（Rizhib）：用来存储入库物料的详细信息。该表的结构如表 6.4 所示。

表 6.4　日志表

字　段　名	数 据 类 型	度	描
Name	varchar	50	用户名
DLsj	varchar	50	登录时间
DZ	varchar	200	动作

☑ 用户表（Users）：用来存储用户的相关信息。该表的结构如表 6.5 所示。

表 6.5　用户表

字　段　名	数 据 类 型	度	描
Username	varchar	50	用户名
Pwd	varchar	30	密码
JB	varchar	2	级别



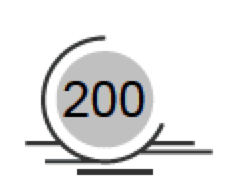
6.4　技 术 准 备

6.4.1　添加 ADO　接类

本实例采用 ADO 来连接 SQL Server 2014 数据库，在使用 ADO 技术时，需要导入一个 ADO 动态链接库 msado15.dll，该动态库位于系统盘下的 Program Files\Common Files\System\ado\目录下。例如，如果系统盘为 C 盘，则该文件位于 C:\Program Files\Common Files\System\ado\目录下。在 Visual C++中，需要使用预处理命令#import，将动态库导入到系统中，代码如下：

```
#import "c:\Program Files\Common Files\System\ado\msado15.dll" no_namespace rename("EOF","adoEOF")
rename ("BOF","adoBOF")
```

添加一个用来连接 ADO 的类。在系统菜单中选择“项目”→“添加类”命令，打开“添加类”对话框，选择“C++类”→“添加”命令，然后输入类名，即完成了类的添加。



创建 ADO 连接类的代码如下：

```
class ADOConn
{
    public:
    _ConnectionPtr m_pConnection;                    //添加一个指向 Connection 对象的指
    _RecordsetPtr m_pRecordset;                      //添加一个指向 Recordset 对象的指
    public:
    ADOConn();
    virtual ~ADOConn();
    void OnInitADOConn();                            //初始化——　接数据库
    _RecordsetPtr& GetRecordSet(_bstr_t bstrSQL);    //执行查询
    BOOL ExecuteSQL(_bstr_t bstrSQL);                //执行 SQL 语句
    void ExitConnect();                              //断开数据库　接
};
```

实现 ADO 连接类函数和程序的代码如下：

```
void ADOConn::OnInitADOConn()
{       //初始化 OLE/COM 库环境
    ::CoInitialize(NULL);
    try {       //创建 connection 对象
            m_pConnection.CreateInstance(__uuidof(Connection));

        //获得　置文件的　径
        TCHAR szFilename[MAX_PATH] = { 0 };
        GetModuleFileName(NULL, szFilename, _countof(szFilename));
        PathRemoveFileSpec(szFilename);
        //读取　置文件
        const PCTSTR szAppName = _T("数据　置");
        CString strFilename = szFilename;
        strFilename += _T("\\Database.ini");

        CString strInitialCatalog, strDataSource, strUserID, strPassword;
        GetPrivateProfileString(szAppName, _T("InitialCatalog"), _T("WenDGL"), strInitialCatalog.GetBuffer
(1024), 1024, strFilename);
        GetPrivateProfileString(szAppName, _T("DataSource"), _T("192.168.1.97"), strDataSource.GetBuffer
(1024), 1024, strFilename);
        GetPrivateProfileString(szAppName, _T("UserID"), _T("sa"), strUserID.GetBuffer(1024), 1024, strFilename);
        GetPrivateProfileString(szAppName, _T("Password"), _T(""), strPassword.GetBuffer(1024), 1024,
strFilename);
        strInitialCatalog.ReleaseBuffer();
        strDataSource.ReleaseBuffer();
        strUserID.ReleaseBuffer();
        strPassword.ReleaseBuffer();

        //设置　接字符串
```

```
            CString str;
            str.Format(_T("Provider=SQLOLEDB.1;Integrated  Security=SSPI;Persist  Security  Info=True;Initial
Catalog=%s;Data Source=%s;User ID=%s;Password=%s;")
                            , strInitialCatalog.GetString()
                            , strDataSource.GetString()
                            , strUserID.GetString()
                            , strPassword.GetString()
                           );
            str.Format(_T("driver={SQL Server};Server=%s;Database=%s;UID=%s;PWD=%s;")
                            , strDataSource.GetString()
                            , strInitialCatalog.GetString()
                            , strUserID.GetString()
                            , strPassword.GetString()
                           );

            _bstr_t strConnect = str;

            //SERVER 和 UID、PWD 的设置根据实  情况来设置
            m_pConnection->Open(strConnect, strUserID.GetString(), strPassword.GetString(), adModeUnknown);
      }
      //捕捉异常
      catch(_com_error e) {
            //显示  误信息
            AfxMessageBox(_T("  接数据库失败!"));
            AfxMessageBox(e.Description());
      }
      catch(...) {
            AfxMessageBox(_T("  接数据库失败!"));
      }
}
_RecordsetPtr& ADOConn::GetRecordSet(_bstr_t bstrSQL)
{
       try {  //  接数据库，如果 Connection 对象为空，则  新  接数据库
                     if(m_pConnection==NULL)            OnInitADOConn();
                     //创建记录  对象
                     m_pRecordset.CreateInstance("ADODB.Recordset");
                     //取得表中的记录
                     m_pRecordset->Open(bstrSQL,m_pConnection.GetInterfacePtr(),adOpenDynamic,
                     adLockOptimistic,adCmdText);
              }
              catch(_com_error e) e.Description();                               //显示  误信息
              return m_pRecordset;                                                //  回记录
}
BOOL ADOConn::ExecuteSQL(_bstr_t bstrSQL)
{
       _variant_t RecordsAffected;
       try {  //是否已  接数据库
                     if(m_pConnection==NULL){
```

```
                OnInitADOConn();
        }
            m_pConnection->Execute(bstrSQL,NULL,adCmdText);
                return true;
            }
        catch(_com_error e)
        {
            e.Description();
            return false;
        }
}
void ADOConn::ExitConnect()
{       //关  记录  和  接
        if(m_pRecordset!=NULL){
            m_pRecordset->Close();
}
        m_pConnection->Close();
        //  放环境
        ::CoUninitialize();
}
```

6.4.2　添加数据库表的类

要利用 ADO 访问数据库，最好对每个表都创建一个类，类的成员变量对应表的列，类的成员函数对应成员变量和表的操作。

为单位表创建新类的代码如下：

```
class CDwxxb
{
        private:
        int DWbh;                                       //单位编号
        CString DWmc;                                   //单位名称
        CString Lxr;                                    //联系人
        CString Lxdh;                                   //联系电话
        CString Lxdz;                                   //联系地址
        CString Memo;                                   //备注信息
        public:
        CDwxxb();
        virtual ~CDwxxb();
        CStringArray a_DWbh;
        CStringArray a_DWmc;
        int GetDWbh();                                  //获得单位编号
        void SetDWbh(int iDWbh);                        //设置单位编号
        CString GetDWmc();                              //获得单位名称
        void SetDWmc(CString cDWmc);                    //设置单位名称
        CString GetLxr();                               //获得联系人
```

```
        void SetLxr(CString cLxr);                         //设置联系人
        CString GetLxdh();                                 //获得联系电话
        void SetLxdh(CString cLxdh);                       //设置联系电话
        CString GetLxdz();                                 //获得联系地址
        void SetLxdz(CString cLxdz);                       //设置联系地址
        CString GetMemo();                                 //获得备注信息
        void SetMemo(CString cMemo);                       //设置备注信息
        void sql_insert();                                 //插入
        void sql_update(int iDWbh);                        //更新
        void sql_delete(int iDWbh);                        //删
        void Load_dep();                                   //批  读取表中数据
        int HaveId(int iDWbh);                             //判断是否存在相同记录
};
```

单位表的类的函数实现代码如下：

```
int CDwxxb::GetDWbh(){ return DWbh; }
void CDwxxb::SetDWbh(int iDWbh) { DWbh=iDWbh; }
CString CDwxxb::GetDWmc(){ return DWmc; }
void CDwxxb::SetDWmc(CString cDWmc) { DWmc=cDWmc; }
CString CDwxxb::GetLxr(){ return Lxr; }
void CDwxxb::SetLxr(CString cLxr) { Lxr=cLxr; }
CString CDwxxb::GetLxdh(){ return Lxdh; }
void CDwxxb::SetLxdh(CString cLxdh) { Lxdh=cLxdh;}
CString CDwxxb::GetLxdz(){return Lxdz; }
void CDwxxb::SetLxdz(CString cLxdz) { Lxdz=cLxdz; }
CString CDwxxb::GetMemo(){return Memo; }
void CDwxxb::SetMemo(CString cMemo) { Memo=cMemo; }
void CDwxxb::sql_insert()                                  //插入一条记录
{
      ADOConn m_AdoConn;
       CString vSQL;
       vSQL.Format("INSERT INTO Dwxxb(DWbh,DWmc,Lxr,Lxdh,Lxdz,Memo)VALUES(%d,'"
       +DWmc+"','"+Lxr+"','"+Lxdh+"','"+Lxdz+"','"+Memo+"')",DWbh); //设置插入语句
       m_AdoConn.ExecuteSQL(_bstr_t(vSQL));                //执行插入语句
      m_AdoConn.ExitConnect();                             //断开数据库  接
}
void CDwxxb::sql_update(int iDWbh)                         //更新一条记录
{
       ADOConn m_AdoConn;
       CString vSQL;
       vSQL.Format("UPDATE Dwxxb SET DWmc='"+DWmc+"',Lxr='"+Lxr+"',Lxdh='"
              +Lxdh+"',Lxdz='"+Lxdz+"',Memo='"+Memo+"' WHERE DWbh=%d",iDWbh);
       m_AdoConn.ExecuteSQL(_bstr_t(vSQL));                //执行修改语句
       m_AdoConn.ExitConnect();                            //关  数据库  接
}
void CDwxxb::sql_delete(int iDWbh)
{
```



```
        ADOConn m_AdoConn;
        m_AdoConn.OnInitADOConn();                              //  接数据库
        CString sql;
        sql.Format("delete from Dwxxb where DWbh='%i'",iDWbh);  //设置删  语句
        m_AdoConn.ExecuteSQL((_bstr_t)sql);                     //执行删  语句
        m_AdoConn.ExitConnect();                                //断开数据库  接
}
void CDwxxb::Load_dep()
{
        ADOConn m_AdoConn;
        m_AdoConn.OnInitADOConn();                              //  接数据库
        _bstr_t vSQL="SELECT*FROM Dwxxb ORDER BY DWbh";         //设置 SQL 语句
        _RecordsetPtr m_pRecordset=m_AdoConn.GetRecordSet(vSQL);
                                                                //初始化数组
        a_DWbh.RemoveAll();                                     //单位编号
        a_DWmc.RemoveAll();                                     //单位名称
        while(m_pRecordset->adoEOF==0)
        {       //获得记录中数据
                a_DWbh.Add((LPCTSTR)(_bstr_t)m_pRecordset->GetCollect("DWbh"));
                a_DWmc.Add((LPCTSTR)(_bstr_t)m_pRecordset->GetCollect("DWmc"));
                m_pRecordset->MoveNext();                       //下一条记录
        }
        m_AdoConn.ExitConnect();                                //断开数据库的  接
}
int CDwxxb::HaveId(int iDWbh)
{
        ADOConn m_AdoConn;
        m_AdoConn.OnInitADOConn();                              //  接数据库
        CString strDWbh;                                        //声明字符串
        _RecordsetPtr m_pRecordset;
        strDWbh.Format("SELECT*FROM Dwxxb WHERE DWbh=%d",iDWbh);
        m_pRecordset=m_AdoConn.GetRecordSet(_bstr_t(strDWbh));  //查询
        return (m_pRecordset->adoEOF)?-1:1;                     //判断是否存在数据
        m_AdoConn.ExitConnect();                                //断开数据库  接
}
```

为类别表创建新类的代码如下：

```
class CZdmlb
{
        private:
        int DWbh;                                               //单位编号
        int LBbh;                                               //类别编号
        CString LBmc;                                           //类别名称
        public:
        CZdmlb();
        virtual ~CZdmlb();
        CStringArray a_DWbh;
```

```
    CStringArray a_LBbh;
    CStringArray a_LBmc;
    int GetDWbh();
    void SetDwbh(int iDWbh);
    int GetLBbh();
    void SetLBbh(int iLBbh);
    CString GetLBmc();
    void SetLBmc(CString cLBmc);
    void sql_insert();                              //插入函数
    void sql_update(int iDWbh,int iLBbh);           //更新函数
    void sql_delete(int iDWbh,int iLBbh);           //删  函数
    void sql_deletedw(int iLBbh);                   //删  单位
    void Load_dep();                                //加  数据
    int HaveId(int iDWbh,int iLBbh);                //判断记录是否存在
};
```

为文档表创建新类的代码如下：

```
class CZdxxb
{
    private:
    int LBbh;                                       //类别编号
    int WDbh;                                       //文档编号
    int DWbh;                                       //单位编号
    CString GJz;                                    //关  字
    CString WDmc;                                   //文档名称
    CString WJlj;                                   //文件  径
    CString Memo;                                   //备注信息
    CString Tjrxm;                                  //添加人姓名
    public:
    CZdxxb();
    virtual ~CZdxxb();
    CStringArray a_WDbh;
    CStringArray a_LBbh;
    CStringArray a_WJlj;
    CStringArray a_DWbh;
    CStringArray a_WDmc;
    CStringArray a_GJz;
    int GetDWbh();
    void SetDWbh(int iDWbh);
    int GetLBbh();
    void SetLBbh(int iLBbh);
    int GetWDbh();
    void SetWDbh(int iWDbh);
    CString GetGJz();
    void SetGJz(CString cGJz);
    CString GetWDmc();
    void SetWDmc(CString cWDmc);
```



```
        CString GetWJlj();
        void SetWJlj(CString cWJlj);
        CString GetMemo();
        void SetMemo(CString cMemo);
        CString GetTjrxm();
        void SetTjrxm(CString cTjrxm);
        void sql_insert();                                  //插入
        void sql_update(int iWDbh);                         //更新
        void sql_deletelb(int iDWbh,int iLBbh);             //删　类别
        void sql_delete(int iWDbh);                         //删　文档
        void sql_deletedw(int iDWbh);                       //删　单位
        int sql_selectwdmc(CString cWDmc);                  //查询文档名称
        void Load_dep();                                    //加　数据
        int HaveId(int iDWbh,int iLBbh,int iWDbh);          //判断文档是否存在
};
```

为日志表创建新类的代码如下：

```
class CRizhib
{
        private:
        CString Name;                                       //用户名
        CString DLsj;                                       //登录时
        CString DZ;                                         //动作
        public:
        CRizhib();
        virtual ~CRizhib();
        CString GetName();
        void SetName(CString cName);
        CString GetDLsj();
        void SetDLsj(CString cDLsj);
        CString GetDZ();
        void SetDZ(CString cDZ);
        void sql_insert();                                  //插入数据
};
```

为用户表创建新类的代码如下：

```
class CUsers
{
        private:
        CString Username;                                   //用户名
        CString Pwd;                                        //密码
        CString JB;                                         //级别
        public:
        CUsers();
        virtual ~CUsers();
        CString GetUsername();
```

```
    void SetUsername(CString cUsername);                    //设置用户名
    CString GetPwd();
    void SetPwd(CString cPwd);                              //设置密码
    CString GetJB();
    void SetJB(CString cJB);                                //设置级别
    void sql_insert();                                      //插入记录
    void sql_update(CString cUsername);                     //更新记录
    void sql_delete(CString cUsername);                     //删  记录
    void sql_updatepwd(CString cUsername);
    int Havename(CString cUsername);                        //判断是否存在相同用户名
    int HaveCzy(CString cUsername,CString cPwd);            //判断用户名、密码是否存在
    //判断用户名、密码和级别是否存在
    int HaveCzyjb(CString cUsername,CString cPwd,CString cJB);
};
```


视频讲解

6.5 主窗体设计

6.5.1 主窗体概

主窗体主要用于对文档管理系统中的各个模块进行调用，并在主模块中显示操作员的姓名及日期，如图 6.5 所示。

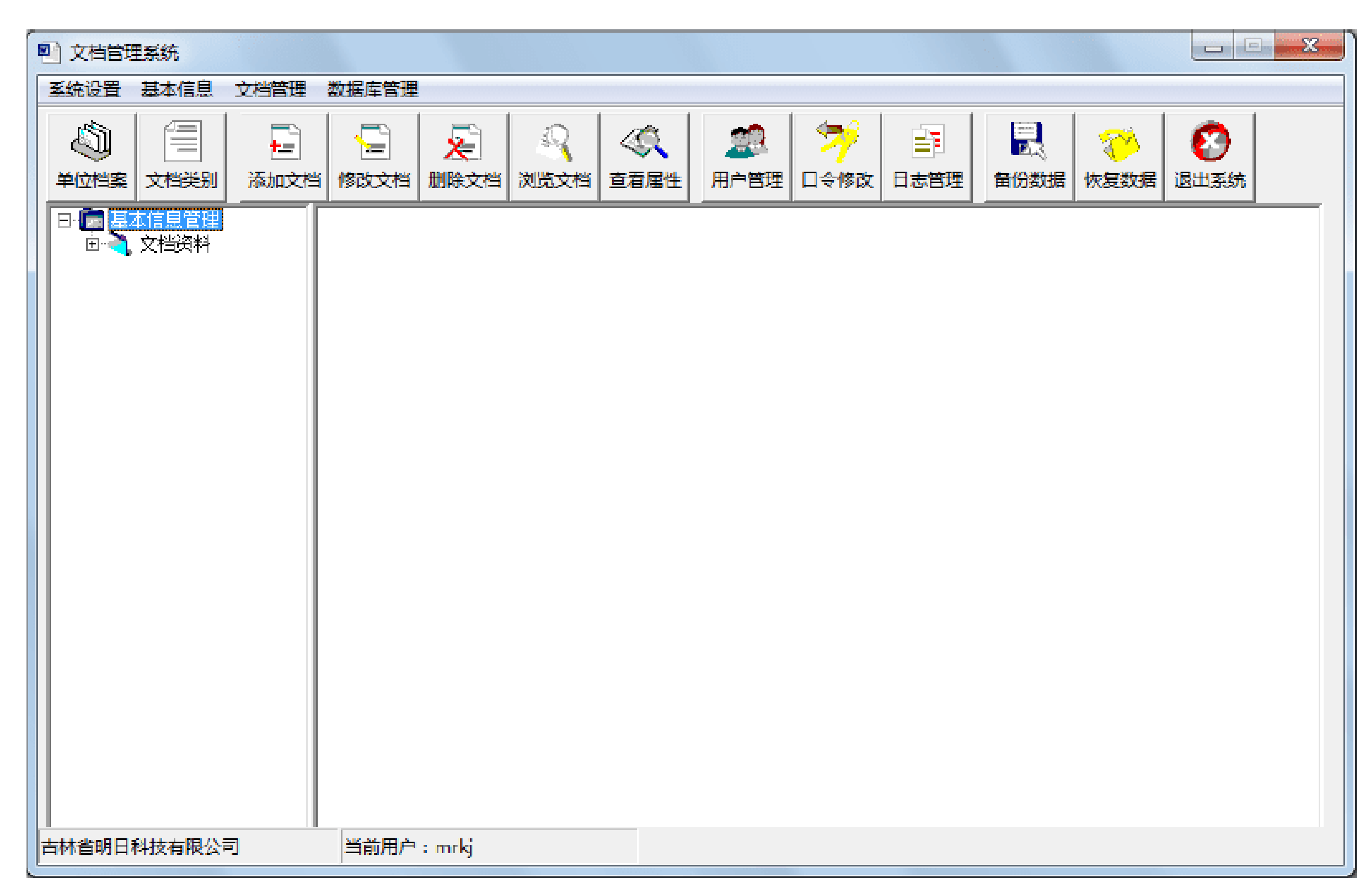
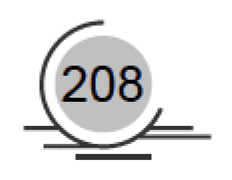


图 6.5 主窗体的运行效果

（1）选择“资源视图”选项卡，在 WordGLXT 选项上右击，在弹出的快捷菜单中选择“添加”→“资源”命令，如图 6.6 所示。打开“添加资源”对话框，如图 6.7 所示。





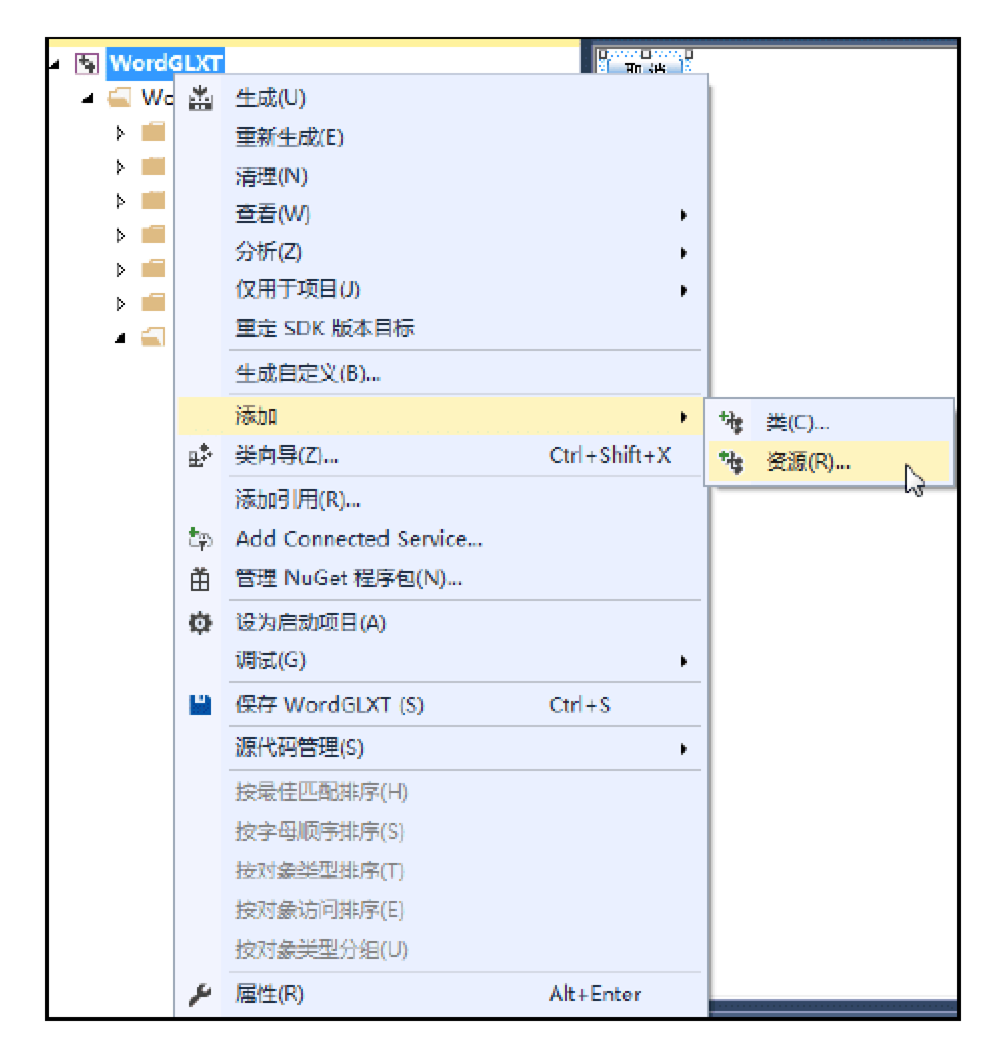


图 6.6　选择“添加”→“资源”命令

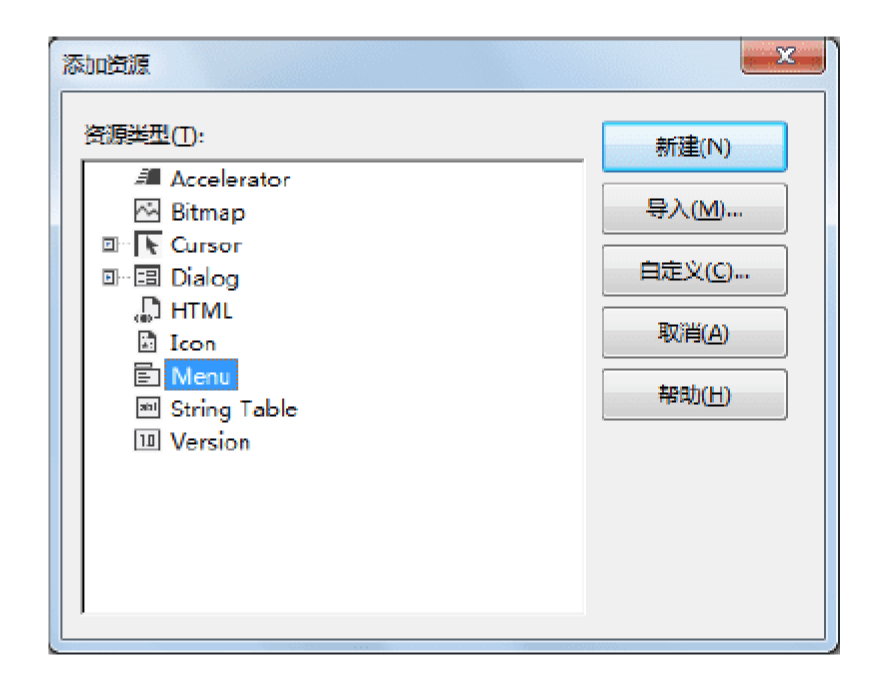


图 6.7　“添加资源”对话框

（2）选择 Meun 文件夹，单击“新建”按钮，WordGLXT 目录下新增一个 Menu 目录项，菜单 ID 为 IDR_MENU1，然后双击此菜单，对此菜单项的属性进行设计，代码如下：

```
IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "系统设置"
    BEGIN
        MENUITEM "用户管理", ID_MENUYHGL
        MENUITEM "口令修改", ID_MENUMODPWD
        MENUITEM SEPARATOR
        MENUITEM "   出系统", ID_System_Exit
    END
    POPUP "基本信息"
    BEGIN
        MENUITEM "单位档案", ID_MENUDWDAN
        MENUITEM SEPARATOR
        MENUITEM "文档类别", ID_MENUWDLB
    END
    POPUP "文档管理"
    BEGIN
        MENUITEM "添加文档", ID_MENUADDWD
        MENUITEM "修改文档", ID_MENUMODWD
        MENUITEM "删   文档", ID_MENUDELWD
        MENUITEM SEPARATOR
        MENUITEM "文档浏览", ID_MENULIULWD
        MENUITEM "查看文件属性", IDD_MENULookFileAttri
    END
    POPUP "数据库管理"
    BEGIN
        MENUITEM "数据库备份", ID_Menu_DBBackUp
```



```
            MENUITEM "数据库恢复", ID_Menu_DBRestore
            MENUITEM "日志管理", ID_MENURZGL
    END
END
```

（3）向主窗体中添加控件，控件的放置位置如图 6.8 所示。

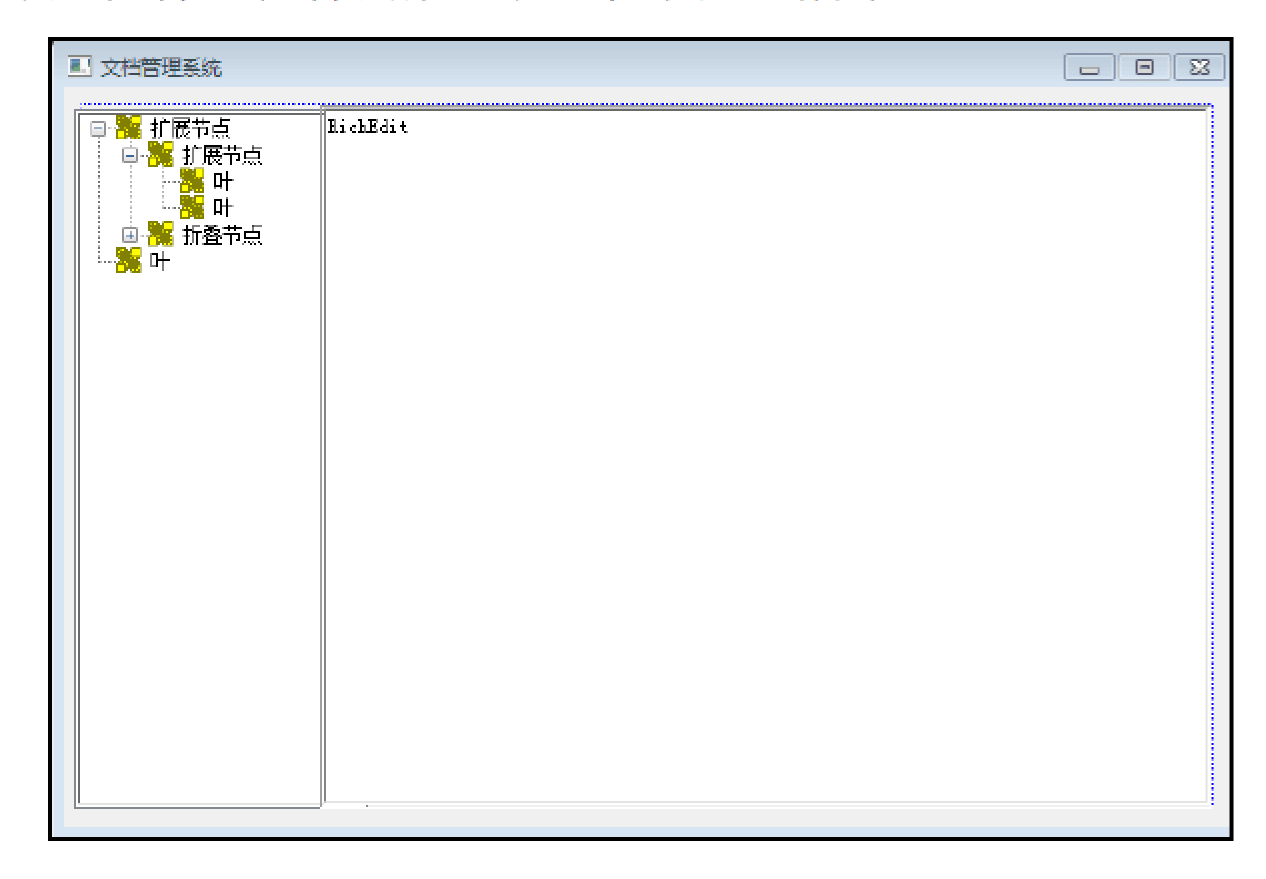


图 6.8　添加控件

（4）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.9 所示。

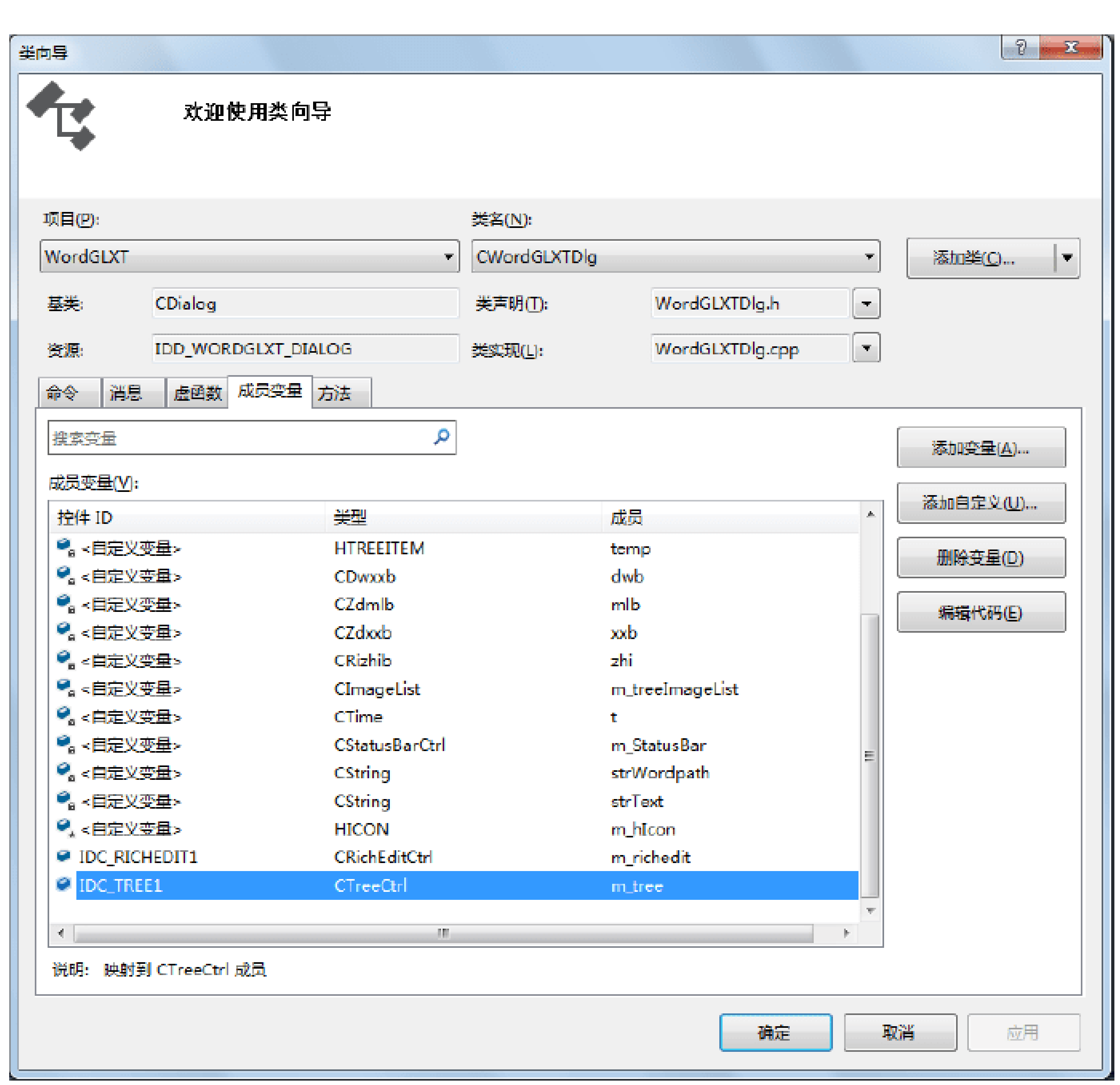


图 6.9　“类向导”对话框



6.5.2　主窗体实现　程

（1）声明 CTime、CStatusBarCtrl 类对象实体，代码如下：

```
CTime t;
CStatusBarCtrl m_StatusBar;                                      //状态栏对象
```

（2）在程序中，引用外部变量的代码如下：

```
extern CWordGLXTApp theApp;
```

（3）在头文件中定义程序变量，代码如下：

```
HTREEITEM arrays[10],brrays[20],hitem[100];
    HTREEITEM m_root,temp;
    CDwxxb dwb;
    CZdmlb mlb;
    CZdxxb xxb;
    CRizhib zhi;
    CImageList m_treeImageList;
    CTime t;
    CStatusBarCtrl m_StatusBar;
    CString strWordpath;                                         //记录 Word　径
    CString strText;                                             //暂存 Word 文档的内容
```

（4）在 OnInitDialog 成员函数中添加状态栏，以及给树形视图控件定义图标、添加数据，代码如下：

```
dwb.Load_dep();
mlb.Load_dep();
xxb.Load_dep();
m_treeImageList.Create(16,16,ILC_MASK,4,1);                      //创建图像列表
m_treeImageList.Add(theApp.LoadIcon(IDI_ROOTICON));
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON1));
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON2));
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON4));
m_tree.SetImageList(&m_treeImageList,LVSIL_NORMAL);
m_root=m_tree.InsertItem("基本信息管理",0,0);                    //插入根节点
AddtoTree(m_root);                                               //向树控件中插入数据
m_tree.Expand(m_root,TVE_EXPAND);                                //展开节点
m_StatusBar.EnableAutomation();
m_StatusBar.Create(WS_CHILD|WS_VISIBLE,CRect(0,0,0,0),this,0);   //创建状态栏
int width[]={200,400};
m_StatusBar.SetParts(4, &width[0]);                              //设置状态栏　板
m_StatusBar.SetText("吉林省明日科技有　公司",0,0);               //设置文本
CString StatusText;
StatusText.Format("当前用户：%s",user.GetUsername());            //显示当前用户
m_StatusBar.SetText(StatusText,0,1);
t=CTime::GetCurrentTime();                                       //获得当前时
```

```
CString strdate;
strdate.Format("当前日期:%s",t.Format("%y-%m-%d"));
m_StatusBar.SetText(strdate,0,2);
return TRUE;
```

（5）定义 AddtoTree 函数，将各表中的数据按层次结构添加到树形视图控件中，其实现代码如下：

```
void CWordGLXTDlg::AddtoTree(HTREEITEM m_node)
{
    int i,j;
❶   for(i=0;i<dwb.a_DWbh.GetSize();i++)
  {
❷       arrays[i]=m_tree.InsertItem(dwb.a_DWmc.GetAt(i),1,1,m_node);
        for(j=0;j<mlb.a_DWbh.GetSize();j++)
        {
            if(atoi(dwb.a_DWbh.GetAt(i))==atoi(mlb.a_DWbh.GetAt(j)))
         {
              brrays[j]=m_tree.InsertItem(mlb.a_LBmc.GetAt(j),2,2,arrays[i]);
         }
    }
}
for(i=0;i<xxb.a_WDbh.GetSize();i++)
{
        for(j=0;j<mlb.a_DWbh.GetSize();j++)
        {
            if(atoi(xxb.a_DWbh.GetAt(i))==atoi(mlb.a_DWbh.GetAt(j))&&atoi  (xxb.a_LBbh.GetAt(i))==
atoi(mlb.a_LBbh.GetAt(j)))
              {
               hitem[i]=m_tree.InsertItem(xxb.a_WDmc.GetAt(i),3,3,brrays[j]);
               }
         }
  }
❸   m_tree.SetRedraw();
}
```

代码贴士

❶ GetSize 函数：返回 CstringArray 对象的长度。
❷ GetAt 函数：返回 CString，从 CstringArray 对象的指定位置读取数据。
❸ SetRedraw 函数：重画树形视图控件。

（6）为树形视图控件添加 OnDblclkTree1 双击事件，其实现代码如下：

```
void CWordGLXTDlg::OnDblclkTree1(NMHDR* pNMHDR, LRESULT* pResult)
{
  CString strWjian="";
  //读取当前节点
❶     temp = m_tree.GetSelectedItem();
  //将当前节点子节点赋给 temp
❷     temp = m_tree.GetChildItem(temp);
```



```
 if (temp != NULL)
 {
      while (temp!= NULL)
      {
             //取出 temp 中的文本
❸            CString strTemp= m_tree.GetItemText(temp);
             strWjian+=strText+"\n";
             //RichEdit 控件显示数据
             m_richedit.SetWindowText(strWjian);
             //将 temp 的兄弟节点赋给 temp
❹            temp = m_tree.GetNextItem(temp,TVGN_NEXT);
      }
 }
 else
 {
      temp = m_tree.GetSelectedItem();
      for(int i=0;i<xxb.a_WDbh.GetSize();i++)
      {
            if(temp==hitem[i])
            {
                  //取出 temp 对应的文档  径
                  strWordpath=xxb.a_WJlj.GetAt(i);
           CFileFind file;
if(!file.FindFile(strWordpath))
{//查找文件是否存在，不存在则清  数据库中的记录
MessageBox("文件不存在！","文档管理系统");
int wdbh=0;
wdbh = atoi(xxb.a_WDbh.GetAt(i));
//删  该文档
UpdateData(true);
xxb.sql_delete(wdbh); //删  该文档
MessageBox("数据库中该文件的记录已删  ！","文档管理系统");
UpdateData(false);
return;
}

             }
}

      //word 应用程序的调用
       _Application app;
       //初始化  接
       //***解决初始化  接时的  程冲突      *****************
       LPDISPATCH   pDisp;
LPUNKNOWN   pUnk;
       CLSID clsid;
       CLSIDFromProgID(L"word.Application",&clsid);
       if(GetActiveObject(clsid,NULL,&pUnk)==S_OK)
       {
```

```
            pUnk->QueryInterface(IID_IDispatch,(void**)&pDisp);
            app.AttachDispatch(pDisp);

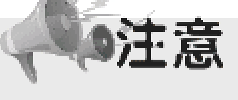
        }
        else
            if(!app.CreateDispatch("word.Application"))     //启动 word
            {
                MessageBox("Word 启动失败！","文档管理系统");
                return;
            }
            //***解决初始化    接时的    程冲突        *****************

            Documents doc;
            CComVariant a (_T(strWordpath)),b(false),c(0),d(true),aa(0),bb(1);
            _Document doc1;

            doc.AttachDispatch( app.GetDocuments());
            doc1.AttachDispatch(doc.Add(&a,&b,&c,&d));
            Range range;

            //取出文档的所    区域
            range = doc1.GetContent();
            //取出文件内容
            strText = range.GetText();
            m_richedit.SetWindowText(strText);
            //关
            app.Quit(&b,&c,&c);
            //    放环境
            range.ReleaseDispatch();
            doc.ReleaseDispatch();
            doc1.ReleaseDispatch();
            app.ReleaseDispatch();

        }
        *pResult = 0;
}
```


代码贴士

❶ GetSelectedItem 函数：获得当前选定树形视图控件的项目。
❷ GetChildItem 函数：获取指定树形视图控件的子项目。
❸ GetItemText 函数：返回项目的文本。
❹ GetNextItem 函数：获取与指定关系相匹配的下一个树形视图控件项目。


注意

需要向程序中添加_Application、Documents、_Document 和 Range 等类。

（7）打开“类向导”对话框，为菜单项 ID_MENULIULWD 添加代码，实现文档浏览功能。其实



现代码如下：

```
void CWordGLXTDlg::OnMenuliulwd()
{
        CString strd,strs;
        for(int i=0;i<xxb.a_WDbh.GetSize();i++)
        {
        strd=xxb.a_WDmc.GetAt(i);                          //获得文档名称
        strs+=strd+"\n";
         m_richedit.SetWindowText(strs);                  //显示文档名称
        }
}
```

（8）为菜单项 ID_MENUADDWD 添加代码，实现添加文档功能。其实现代码如下：

```
void CWordGLXTDlg::OnMenuaddwd()                          //实现添加文档功能
{
        CWDgldlg dlg;
        dlg.str = 0;
        if(dlg.DoModal()==IDOK)
        {
                m_tree.DeleteAllItems();
                dwb.Load_dep();
                mlb.Load_dep();
                xxb.Load_dep();
                m_root=m_tree.InsertItem("基本信息管理",0,0);
                AddtoTree(m_root);
                }
}
```

（9）为菜单项 ID_MENUDELWD 添加代码，实现删除文档功能。其实现代码如下：

```
void CWordGLXTDlg::OnMenudelwd()                          //实现删  文档功能
{
        CWDgldlg dlg;
        dlg.tabindex = 1;
        if(dlg.DoModal()==IDOK)
        {
                m_tree.DeleteAllItems();
                dwb.Load_dep();
                mlb.Load_dep();
                xxb.Load_dep();
                m_root=m_tree.InsertItem("基本信息管理",0,0);
                AddtoTree(m_root);
        }
}
```

（10）为菜单项 ID_MENURZGL 添加代码，实现日志管理功能。其实现代码如下：



```
void CWordGLXTDlg::OnMenurzgl()
{
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();                                  //   接数据库
 CString sql,sqlzd="用户名\t 登录时   \t 动作\n";
 sql.Format("select* from Rizhib");                          //设置查询语句
 m_AdoConn.GetRecordSet((_bstr_t)sql);                       //查询日志信息
 while(m_AdoConn.m_pRecordset->adoEOF== 0)
 {
     //获得用户名
      sqlzd+=(char*)(_bstr_t)m_AdoConn.m_pRecordset->GetCollect("name");
      sqlzd+="   \t";
     //获得登录时
      sqlzd+=(char*)(_bstr_t)m_AdoConn.m_pRecordset->GetCollect("DLsj");
      sqlzd+="\t";
     //获得动作
      sqlzd+=(char*)(_bstr_t)m_AdoConn.m_pRecordset->GetCollect("dz");
      sqlzd+="\n";
      m_AdoConn.m_pRecordset->MoveNext();                    //移动到下一条记录
      m_richedit.SetWindowText(sqlzd);
 }
 m_AdoConn.ExitConnect();
}
```

（11）为菜单项 ID_EXIT 添加代码，程序调用 OnOK 函数关闭对话框，退出系统，代码如下：

```
void CWordGLXTDlg::OnExit()
{
     OnOK();
}
```



6.6 登录管理模块设计

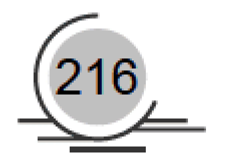
6.6.1 登录管理模块概

登录管理模块主要用于对登录文档管理系统的用户进行安全性检查，以防止非法用户进入该系统。只有合法的用户才可以登录系统，同时根据操作员的不同给予其相应的操作权限。

验证操作员及其密码主要是通过对用户表的查询，结合 if 语句判断用户选定的操作员及其输入的密码是否符合数据库中的操作员和密码，如果符合则允许登录，并给予相应的权限，否则提示错误信息。文档管理系统的“登录管理”界面如图 6.10 所示。



图 6.10　“登录管理”界面





6.6.2　登录管理模块技术分析

为了保证用户在某控件上按 Enter 键时，都会将焦点定位到下一个控件上，可以通过将 Enter 键转换成 Tab 键来实现。在本模块中系统就采取了这种方法，输入用户名后按 Enter 键，焦点就会移动到“密码”文本框中，实现代码如下：

```
BOOL CYHgldlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message==WM_KEYDOWN && pMsg->wParam==13)
        pMsg->wParam=9;
    return CDialog::PreTranslateMessage(pMsg);
}
```

6.6.3　登录管理模块实现　程

登录管理模块实现过程如下：

（1）在工程中新建一个对话框，将对话框的 ID 设为 IDD_DIALOGIN。

（2）向对话框中导入 4 个位图资源。

（3）向对话框中添加一个图像控件和两个文本框控件，各控件的摆放位置如图 6.11 所示。

（4）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.12 所示。



图 6.11　“登录管理”界面

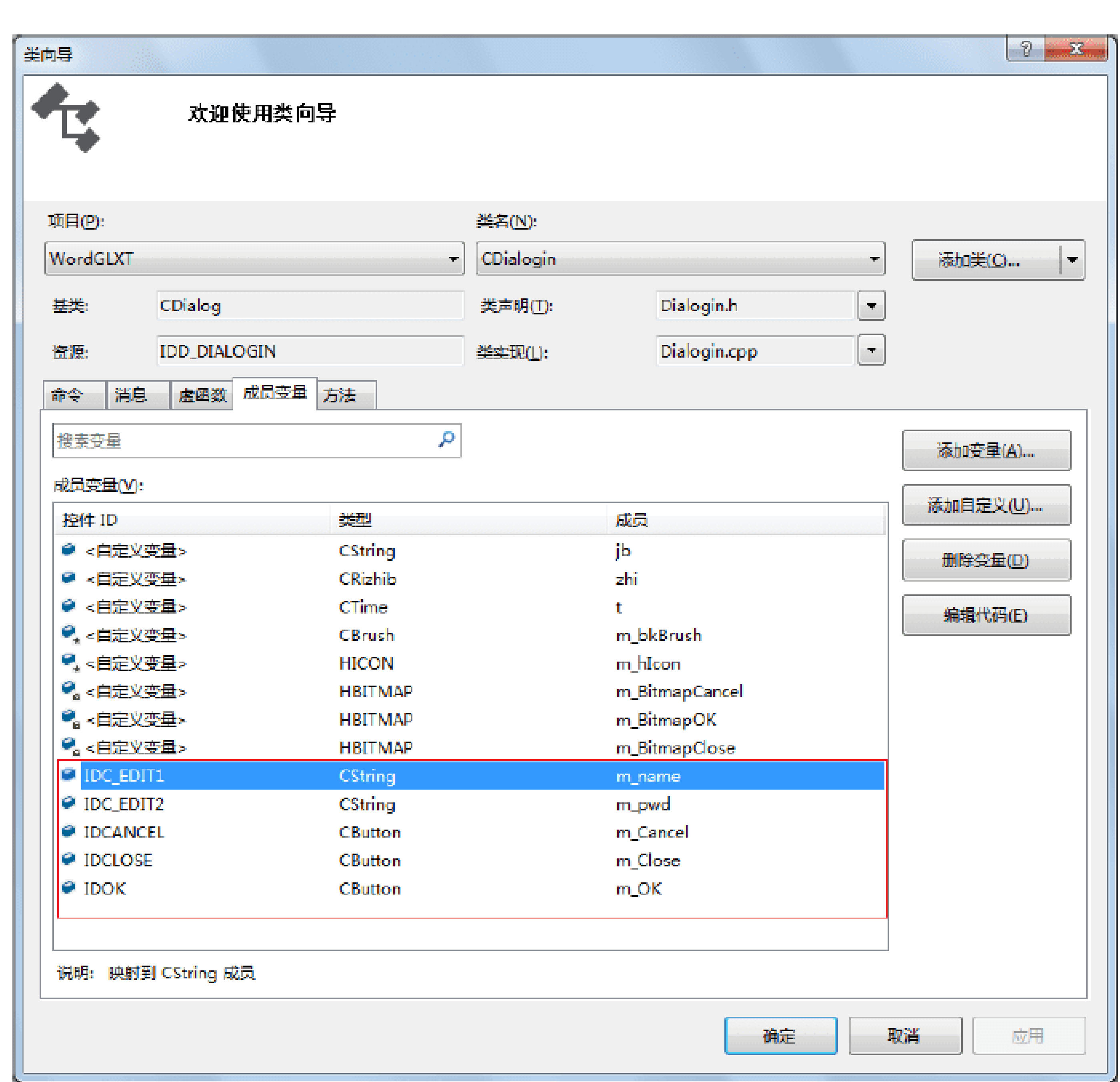


图 6.12　“类向导”对话框

（5）引用函数外部的变量的代码如下：

```
extern CUsers user;
```

在头文件中定义程序变量的代码如下：

```
CString jb;
CRizhib zhi;
CTime t;
```

为了使登录界面美观，还需要更换按钮的界面，代码如下：

```
BOOL CDialogin::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetIcon(m_hIcon, TRUE);
	m_BitmapOK = ::LoadBitmap(::AfxGetInstanceHandle(), MAKEINTRESOURCE
(IDB_BITMAP_QR));                                        //为"登录"按  加  位图
		m_BitmapCancel = ::LoadBitmap(::AfxGetInstanceHandle(), MAKEINTRESOURCE
(IDB_BITMAP_QX));                                        //为"取消"按  加  位图

	m_BitmapClose = ::LoadBitmap(::AfxGetInstanceHandle(), MAKEINTRESOURCE
(IDB_BITMAP_Close));                                     //为窗体的关  按  加  位图
m_OK.SetBitmap(m_BitmapOK);
	m_Cancel.SetBitmap(m_BitmapCancel);
m_Close.SetBitmap(m_BitmapClose);
	return TRUE;
}
```

因为登录窗体的属性中选择隐藏标题栏，所以不能拖动窗体。要实现能拖动窗体，需要加入如下代码：

```
void CDialogin::OnLButtonDown(UINT nFlags, CPoint point)
{	//该函数实现在客户区能够拖动窗体
	CDialog::OnLButtonDown(nFlags, point);
❶	PostMessage(WM_NCLBUTTONDOWN,HTCAPTION,MAKELPARAM(point.x,point.y));
}
```

代码贴士

❶ PostMessage 函数：传送消息函数。

响应“登录”按钮的程序代码如下：

```
void CDialogin::OnOK()
{
	UpdateData(true);                         //将对话框中文本框的数据读取到成员变  中
	if(m_name=="")                            //检查数据有效性
	{
MessageBox("请  入用户名","文档管理系统");
```



```
        return;
}
        if(m_pwd=="")
        {
                MessageBox("请　入密码");
                return;
        }
  if(user.HaveCzy(m_name,m_pwd)!=1)                    //如果读取数据和用户　入的不同，则　回
  {
          MessageBox("用户名或密码　误!","文档管理系统");
          return;
  }
  user.SetUsername(m_name);
  //判断用户级别
  jb="1";
  if(user.HaveCzyjb(m_name,m_pwd,jb)==1)
  {
        user.SetJB(jb);
  }
  else
  {
        user.SetJB("0");
  }
  //读取当前系统时
  t=CTime::GetCurrentTime();
  //将登录动作记录到日志表中
  zhi.SetDLsj(t.Format("%y-%m-%d"));
  zhi.SetName(user.GetUsername());
  zhi.SetDZ("登录");
  zhi.sql_insert();
  CDialog::OnOK();
}
```

注意 为了在该类中使用用户表和日志表的类，需要在头文件中加入用户表和日志表的头文件。

下面在主对话框中添加代码，使对话框在启动时首先打开登录对话框。在主窗口选择 OnInitDialog 函数，该函数将打开“登录”对话框，如果用户不是通过单击“登录”按钮关闭对话框的，则调用 OnOK 函数关闭主对话框，具体代码如下：

```
BOOL CWordGLXTDlg::OnInitDialog()
{

        CDialog::OnInitDialog();

        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}

SetIcon(m_hIcon, TRUE);                                  //设置大图标
SetIcon(m_hIcon, FALSE);                                 //设置小图标

CDialogin gin;

if(gin.DoModal()!=IDOK)                                  //启动登录对话框
OnOK();
dwb.Load_dep();                                          //批  加  单位表中的记录数据
mlb.Load_dep();                                          //批  加  类别表中的记录数据
xxb.Load_dep();                                          //批  加  文档表中的记录数据
m_treeImageList.Create(16,16,ILC_MASK,4,1);
m_treeImageList.Add(theApp.LoadIcon(IDI_ROOTICON));      //显示根目录图标
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON1));    //显示一级目录的图标
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON2));    //显示二级目录的图标
m_treeImageList.Add(theApp.LoadIcon(IDI_CHILDICON4));    //显示三级目录的图标，就是 Word 图标
m_tree.SetImageList(&m_treeImageList,LVSIL_NORMAL);
m_root=m_tree.InsertItem("基本信息管理",0,0);
AddtoTree(m_root);
m_tree.Expand(m_root,TVE_EXPAND);
//状态栏显示内容的设置
m_StatusBar.EnableAutomation();
m_StatusBar.Create(WS_CHILD|WS_VISIBLE,CRect(0,0,0,0),this,0);

int width[]={200,400};
m_StatusBar.SetParts(4, &width[0]);
m_StatusBar.SetText("吉林省明日科技有  公司",0,0);        //显示单位名称

CString StatusText;
StatusText.Format("当前用户：%s",user.GetUsername());     //显示当前用户
m_StatusBar.SetText(StatusText,0,1);

t=CTime::GetCurrentTime();
CString strdate;
strdate.Format("当前日期:%s",t.Format("%y-%m-%d"));       //显示当前时
m_StatusBar.SetText(strdate,0,2);
//工具栏显示内容的设置
m_ImageList.Create(32,32,ILC_COLOR|ILC_MASK,1,1);         //创建图像列表
```



```
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONDWDA));        //单位档案
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONWDLB));        //文档类别

    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONAdd));         //添加文档
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONMod));         //修改文档
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONDel));         //删　文档
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONScan));        //浏览文档
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONFileAttri));   //查看属性
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONUser));        //用户管理
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONMIMA));        //口令修改
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONLog));         //日志管理
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONSJKBF));       //备份
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONSJKHF));       //恢复
    m_ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICONExit));        //　出系统


    UINT array[16];
    for(int i=0;i<16;i++)
    {
        if(i==2||i==8||i==12)
        {
            array[i]=ID_SEPARATOR;                                //第 3 个和第 9 个按　为分　条
        }
        else   array[i]=i+1101;
    }

    m_ToolBar.Create(this);
    m_ToolBar.SetButtons(array,16);
    m_ToolBar.SetButtonText(0,"单位档案");
    m_ToolBar.SetButtonText(1,"文档类别");
    m_ToolBar.SetButtonText(3,"添加文档");
    m_ToolBar.SetButtonText(4,"修改文档");
    m_ToolBar.SetButtonText(5,"删　文档");
    m_ToolBar.SetButtonText(6,"浏览文档");
    m_ToolBar.SetButtonText(7,"查看属性");
    m_ToolBar.SetButtonText(9,"用户管理");
    m_ToolBar.SetButtonText(10,"口令修改");
    m_ToolBar.SetButtonText(11,"日志管理");
    m_ToolBar.SetButtonText(13,"备份数据");
    m_ToolBar.SetButtonText(14,"恢复数据");
    m_ToolBar.SetButtonText(15,"　出系统");
    m_ToolBar.GetToolBarCtrl().SetImageList(&m_ImageList);       //关联图像列表

    m_ToolBar.SetSizes(CSize(60,60),CSize(32,32));                //设置按　和按　位图大小
    m_ToolBar.EnableToolTips(true);
    RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST, 0);//显示工具栏
    ::GetCurrentDirectory(512,buf);                               //获得当前　径，备份数据库时用到
    return TRUE;
}
```



6.7　单位档案模块设计

6.7.1　单位档案模块概

单位档案模块用于查看、添加、修改和删除单位信息，如图 6.13 所示。

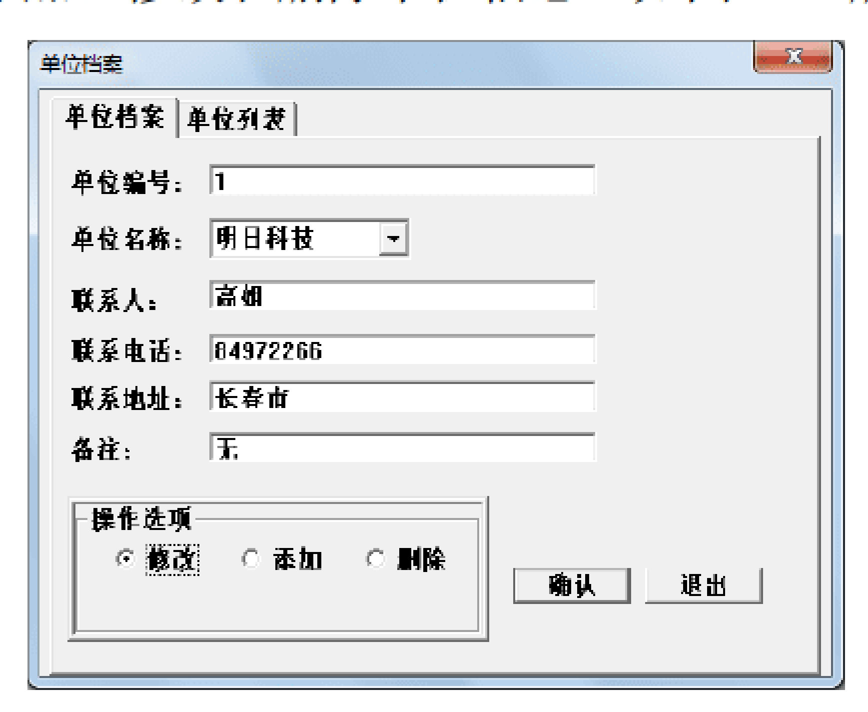


图 6.13　单位档案模块的运行效果

若要添加单位，“单位编号”会默认自动增加。若修改或删除单位档案信息，可以通过“单位名称”下拉列表框选择单位名称，然后修改其内容。也可从“单位列表”选项卡中选择要修改或删除的单位，操作非常方便。

6.7.2　单位档案模块技术分析

在实现单位档案模块时，需要注意以下几点技术细节。

1．TC_ITEM 结构

一个标签可能含有文本、图片等属性，TC_ITEM 结构中的 mask 指定了标签的哪种属性是有效的：mask=TCIF_TEXT，表示文本有效；mask=TCIF_IMAGE，表示图片有效。

2．InsertItem 函数

InsertItem 函数用于插入标签，其语法如下：

```
InsertItem([索引 i]，[TC_ITEM 指   ])
```

☑　索引：插入标签的位置。
☑　TC_ITEM 指针：取得标签的属性。

6.7.3　单位档案模块实现　程

单位档案模块的实现过程如下：



（1）在工程中新建一个对话框，将对话框的 ID 设为 IDD_DWDAN。

（2）向对话框中添加控件资源。各控件的摆放位置如图 6.14 所示。

（3）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.15 所示。



图 6.14　“单位档案”对话框

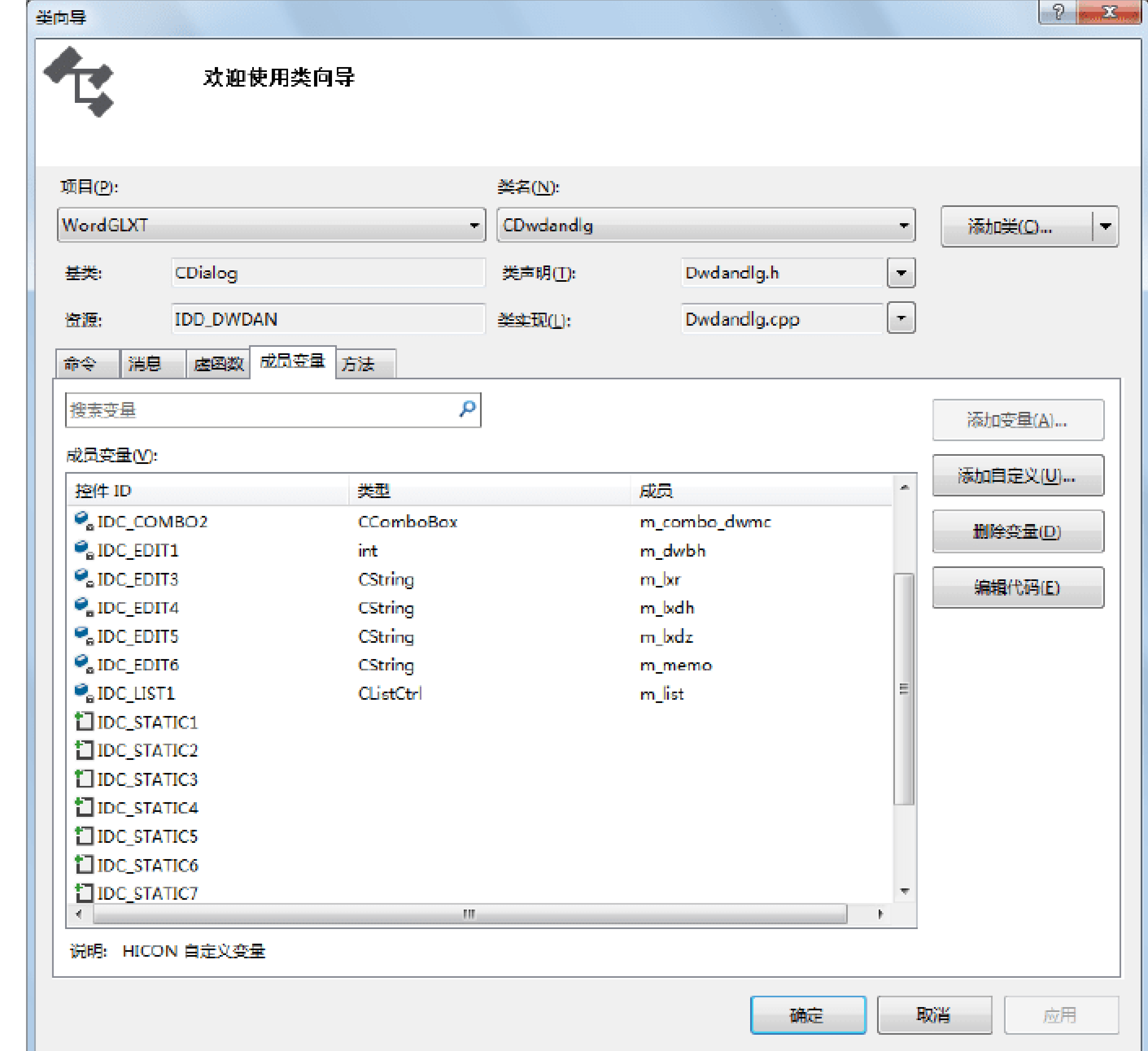


图 6.15　“类向导”对话框

（4）使用函数外部的变量的代码如下：

```
extern CUsers user;
```

在头文件中定义变量的代码如下：

```
CRizhib zhi;
CTime t;
```

在“类向导”对话框中添加 OnInitDialog 函数，此函数用于初始化 Tab 控件，为 ListControl 控件赋值，代码如下：

```
BOOL CDwdandlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetIcon(m_hIcon, TRUE);
	t=CTime::GetCurrentTime();
	TC_ITEM tci;
	tci.mask=TCIF_TEXT;
```

```
    tci.pszText="单位档案";
    m_tab.InsertItem(0,&tci);
    tci.pszText="单位列表";
    m_tab.InsertItem(1,&tci);
    //使 ListControl 控件不可见
    GetDlgItem(IDC_LIST1)->ShowWindow(SW_HIDE);
    UpdateData(true);
    //为 ListControl 控件设置列
    m_list.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|
LVS_EX_GRIDLINES);
    m_list.InsertColumn(0,"单位编号",LVCFMT_LEFT,100,0);
    m_list.InsertColumn(1,"单位名称",LVCFMT_LEFT,100,1);
    m_list.InsertColumn(2,"联系人",LVCFMT_LEFT,100,2);
    m_list.InsertColumn(3,"联系电话",LVCFMT_LEFT,100,3);
    m_list.InsertColumn(4,"联系地址",LVCFMT_LEFT,100,4);
    m_list.InsertColumn(7,"备注",LVCFMT_LEFT,100,5);
    ADOConn m_AdoConn;
    m_AdoConn.OnInitADOConn();                              //   接数据库
    CString sql;                                            //声明字符串
    sql.Format("select* from Dwxxb order by dwbh desc");    //设置查询语句
    m_AdoConn.GetRecordSet((_bstr_t)sql);                   //执行查询
    while(m_AdoConn.m_pRecordset->adoEOF==0)
    {
        m_list.InsertItem(0,"");                            //插入行
                                                            //插入列信息
        m_list.SetItemText(0,0,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("dwbh"));
        m_list.SetItemText(0,1,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("dwmc"));
        m_combo_dwmc.AddString((char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("dwmc"));
        m_list.SetItemText(0,2,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("lxr"));
        m_list.SetItemText(0,3,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("lxdh"));
        m_list.SetItemText(0,4,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("lxdz"));
        m_list.SetItemText(0,5,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("memo"));
        m_AdoConn.m_pRecordset->MoveNext();                 //移动到下一条记录
    }
    m_AdoConn.ExitConnect();                                //关   数据库   接
    CDwxxb dwb;
    dwb.Load_dep();
    m_dwbh = dwb.a_DWbh.GetSize()+1;                        //设置单位档案中   认单位编号
    UpdateData(false);
    return TRUE;
}
```

为 Radio 控件添加消息响应函数，代码如下：



```
void CDwdandlg::OnRADIOModify()
{
	GetDlgItem(IDC_EDIT1)->EnableWindow(TRUE);
	RadioFlag = 1;
}
void CDwdandlg::OnRadioAdd()
{
	CDwxxb dwb;
	dwb.Load_dep();                                        //加　单位数据
	m_dwbh=dwb.a_DWbh.GetSize()+1;                         //　新显示　认的单位编号
	GetDlgItem(IDC_EDIT1)->EnableWindow(FALSE);            //使控件不可用
	RadioFlag = 2;
	UpdateData(false);
}
void CDwdandlg::OnRADIODel()
{
	RadioFlag = 3;
}
```

当用户选中“添加”单选按钮时，能够实现添加单位功能，代码如下：

```
void CDwdandlg::AddDW()                                    //添加单位信息
{
	UpdateData(true);
	CString strdwmc;
	m_combo_dwmc.GetWindowText(strdwmc);                   //获得单位名称
	CDwxxb dwb;
	if(strdwmc=="")
		::AfxMessageBox("单位名称不能为空");
	else if(dwb.HaveId(m_dwbh)==1)
		MessageBox("单位编号已存在","文档管理系统");
	else if(m_lxdh.GetLength()>12)
		MessageBox("电话号码不正确","文档管理系统");
	else
	{
		//设置单位信息
		dwb.SetDWbh(m_dwbh);
		dwb.SetDWmc(strdwmc);
		dwb.SetLxr(m_lxr);
		dwb.SetLxdh(m_lxdh);
		dwb.SetLxdz(m_lxdz);
		dwb.SetMemo(m_memo);
		dwb.sql_insert();
		zhi.SetDLsj(t.Format("%y-%m-%d"));
		zhi.SetName(user.GetUsername());
		zhi.SetDZ("单位添加");
		zhi.sql_insert();
	}
}
```

当用户选中“修改”单选按钮时，实现修改单位功能，代码如下：

```
void CDwdandlg::ModifyDW()                                    //修改单位信息
{
    UpdateData(true);
    CString strdwmc;
    if(m_combo_dwmc.GetCurSel()!=CB_ERR)                      //若从下拉列表框中  择
        m_combo_dwmc.GetLBText(m_combo_dwmc.GetCurSel(),strdwmc);
    else                                                      //若没有从下拉列表框中  择
    {
        m_combo_dwmc.GetWindowText(strdwmc);
        int a =m_combo_dwmc.SelectString(-1,strdwmc);
        if(a == CB_ERR) strdwmc="";                           //若数据库中没有该单位，则清空 strdwmc
    }
    if(strdwmc =="") MessageBox("单位名称不能为空","文档管理系统");
❶      else if(m_lxdh.GetLength()>=12) MessageBox("电话号码不正确","文档管理系统");
    else
    {
        //设置单位信息
        CDwxxb dwb;
        dwb.SetDWmc(strdwmc);
        dwb.SetLxr(m_lxr);
        dwb.SetLxdh(m_lxdh);
        dwb.SetLxdz(m_lxdz);
        dwb.SetMemo(m_memo);
        dwb.sql_update(m_dwbh);
        zhi.SetDLsj(t.Format("%y-%m-%d"));
        zhi.SetName(user.GetUsername());
        zhi.SetDZ("单位修改");
        zhi.sql_insert();
    }
}
```

代码贴士

❶ GetLength 函数：返回 int，取得字符串长度。

当用户选中“删除”单选按钮时，实现删除单位功能，代码如下：

```
void CDwdandlg::DelDW()                                       //删  单位信息
{     CDwxxb dwb;
if(dwb.HaveId(m_dwbh)==-1)
     MessageBox("单位编号不存在,无法执行删  操作!","文档管理系统");
     else
     {
          dwb.sql_delete(m_dwbh);                             //删  单位表中该单位信息
          CZdmlb mlb;
          mlb.sql_deletedw(m_dwbh);                           //删  类别表中该单位信息
          CZdxxb xxb;
          xxb.sql_deletedw(m_dwbh);                           //删  文档表中该单位信息
```

```
            zhi.SetDLsj(t.Format("%y-%m-%d"));
            zhi.SetName(user.GetUsername());
            zhi.SetDZ("单位删　");
            zhi.sql_insert();
        }
}
```

下面要实现标签的切换功能。标签控件有两个重要的消息：TCN_SELCHANGING 和 TCN_SELCHANG。它们分别是在改变当前标签前和选择了新的标签后发出的，这样就可以在 TCN_SELCHANGING 消息响应函数中将原来的控件隐藏，而在 TCN_SELCHANG 消息响应函数中显示新控件。利用 ClassWizard 建立 IDC_TAB1 的 TCN_SELCHANGING 和 TCN_SELCHANG 消息响应函数，并将以下代码加入两个函数中。

```
void CDwdandlg::OnSelchangeTab1(NMHDR* pNMHDR, LRESULT* pResult)
{
❶    switch(m_tab.GetCurSel())
 {
 case 0:                                    //显示控件
❷        GetDlgItem(IDC_EDIT1)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_COMBO2)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_EDIT3)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_EDIT4)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_EDIT5)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_EDIT7)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC1)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC2)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC3)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC4)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC5)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC7)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_STATIC7)->ShowWindow(SW_SHOW);
         GetDlgItem(IDOK)->ShowWindow(SW_SHOW);
         GetDlgItem(IDCANCEL)->ShowWindow(SW_SHOW);

         break;
 case 1:
         GetDlgItem(IDC_LIST1)->ShowWindow(SW_SHOW);
         break;
 }
 *pResult = 0;
}
```

代码贴士

❶ GetCurSel 函数：在标签控件中确定当前选定的标签。

❷ GetDlgItem 函数：获得当前控件的句柄。

```
void CDwdandlg::OnSelchangingTab1(NMHDR* pNMHDR, LRESULT* pResult)
{
```

```
switch(m_tab.GetCurSel())
 {
 case 0:                                                            //  藏控件
        GetDlgItem(IDC_EDIT1)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_COMBO2)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT3)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT4)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT5)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT6)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC1)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC2)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC3)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC4)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC5)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC6)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC7)->ShowWindow(SW_HIDE);
        GetDlgItem(IDOK)->ShowWindow(SW_HIDE);
        GetDlgItem(IDCANCEL)->ShowWindow(SW_HIDE);
        break;
 case 1:
        GetDlgItem(IDC_LIST1)->ShowWindow(SW_HIDE);
        break;
 }
 *pResult = 0;
}
```

当在“单位名称”下拉列表框中选择单位名称时，其他编辑框中的内容会自动显示，完成此功能的代码如下：

```
void CDwdandlg::OnSelchangeCombo2()
{
       GetDlgItem(IDC_EDIT1)->EnableWindow(false);
       CDwxxb dwb;
       CString strdwmc;
       m_combo_dwmc.GetLBText(m_combo_dwmc.GetCurSel(),strdwmc);     //获得单位名称
       dwb.Load_dep();
       int m =dwb.a_DWbh.GetSize();
       for(int i=0;i<m;i++)                                          //根据单位编号搜索单位名称
       {
              if(strdwmc==dwb.a_DWmc.GetAt(i))
              {
                     m_dwbh = atoi(dwb.a_DWbh.GetAt(i));

              }
       }
       UpdateData(false);
}
```





6.8　文档类别模块设计



6.8.1　文档类别模块概

文档类别模块用于添加、修改和删除文档类别信息，“文档类别”对话框的运行效果如图 6.16 所示。

可以通过下拉列表框选择单位，该单位所对应的“单位编号”会自动显示在对应的编辑框中，类别编号会自动增加，操作非常简便。

图 6.16　“文档类别”对话框

6.8.2　文档类别模块实现　程

（1）在工程中新建一个对话框，将对话框的 ID 设为 IDD_WDLB。

（2）向对话框中添加控件资源。各控件的摆放位置如图 6.17 所示。

（3）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.18 所示。

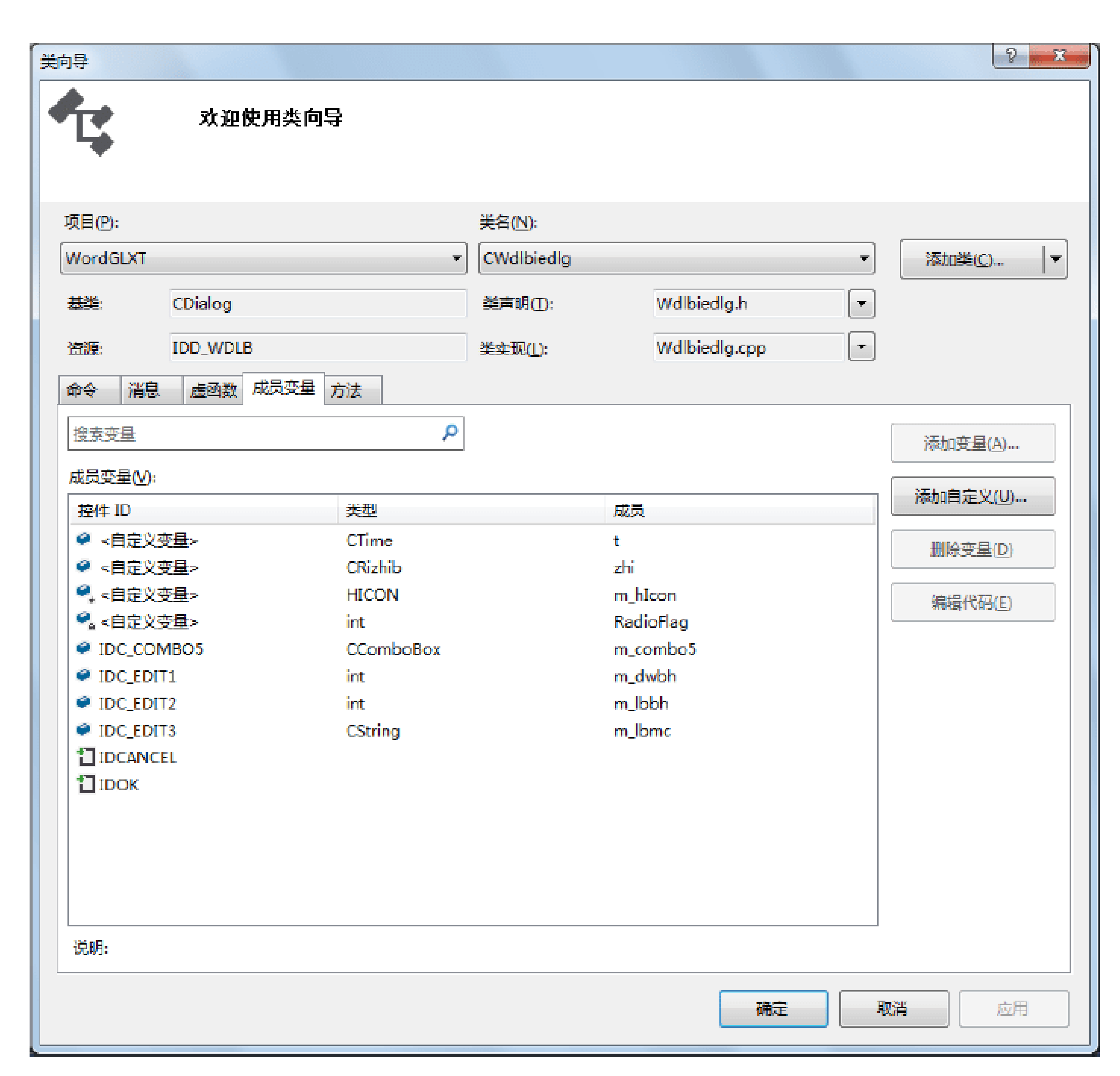


图 6.17　“文档类别”对话框　　图 6.18　“类向导”对话框



（4）使用函数外部的变量的代码如下：

```
extern CUsers user;
```

在头文件中定义变量的代码如下：

```
CRizhib zhi;
CTime t;
```

当用户单击“确认”按钮时，实现对文档类别的指定操作，代码如下：

```
void CWdlbiedlg::OnOK()                                        // “确认”按
{
        switch(RadioFlag)
        {
        case 1: LBModify(); break;                             //修改
        case 2: LBAdd(); break;                                //添加
        case 3: LBDel(); break;                                //删
        default :
                ::AfxMessageBox("请    择操作        !");
                return;
        }
        CDialog::OnOK();
}
```

当用户选中“添加”单选按钮时，实现添加文档类别的功能，代码如下：

```
void CWdlbiedlg::LBAdd()
{
       UpdateData(true);
       if(m_lbmc=="")                                          //判断类别名称是否为空
       {
              MessageBox("类别名称不能为空","文档管理系统");
              return;
       }
       CZdmlb mlb;
CDwxxb dwb;
mlb.Load_dep();
       dwb.Load_dep();
       int dw=0;
       for(int i=0;i<dwb.a_DWbh.GetSize();i++)
       {
              if(m_dwbh==atoi(dwb.a_DWbh.GetAt(i)))            //获得单位编号
              {
                     dw++;
              }
       }
       if(dw==0)
       {
              MessageBox("单位编号不存在","文档管理系统");
```



```
        return;
    }
    dw=0;
    if(mlb.HaveId(m_dwbh,m_lbbh)==1)                              //判断类别是否存在
    {
        MessageBox("类别已存在","文档管理系统");
        return;
    }
    mlb.SetDwbh(m_dwbh);                                          //设置单位编号
    mlb.SetLBbh(m_lbbh);                                          //设置类别编号
    mlb.SetLBmc(m_lbmc);                                          //设置类别名称
    mlb.sql_insert();                                             //插入数据
    zhi.SetDLsj(t.Format("%y-%m-%d"));                            //设置登录时
    zhi.SetName(user.GetUsername());                              //设置用户名
    zhi.SetDZ("类别添加");                                         //设置动作
    zhi.sql_insert();                                             //插入日志
    }
```

当用户选中“修改”单选按钮时，实现修改文档类别的功能，代码如下：

```
void CWdlbiedlg::LBModify()
{
    UpdateData(true);

if(m_lbmc=="")
     {
         MessageBox("类别名称不能为空","文档管理系统");
         return;
     }
     CZdmlb mlb;
     CDwxxb dwb;
     dwb.Load_dep();
     mlb.Load_dep();
     int dw=0;
     for(int i=0;i<dwb.a_DWbh.GetSize();i++)
     {
         if(m_dwbh==atoi(dwb.a_DWbh.GetAt(i)))
         {
             dw++;
         }
     }
     if(dw==0)
     {
         MessageBox("单位编号不存在","文档管理系统");
         return;
     }
     dw=0;

    mlb.SetDwbh(m_dwbh);                                          //设置单位编号
    mlb.SetLBmc(m_lbmc);                                          //设置类别名称
```

```
        mlb.sql_update(m_dwbh,m_lbbh);                        //修改类别
        zhi.SetDLsj(t.Format("%y-%m-%d"));                    //设置登录时
        zhi.SetName(user.GetUsername());                      //设置用户名
        zhi.SetDZ("类别修改");                                 //设置动作
        zhi.sql_insert();                                     //插入日志

}
```

当用户选中“删除”单选按钮时，实现删除文档类别的功能，代码如下：

```
void CWdlbiedlg::LBDel()
{
        UpdateData(true);
        CZdmlb mlb;
        mlb.sql_delete(m_dwbh,m_lbbh);                        //删　类别
        CZdxxb xxb;
        xxb.sql_deletelb(m_dwbh,m_lbbh);                      //删　文档
        zhi.SetDLsj(t.Format("%y-%m-%d"));                    //设置登录时
        zhi.SetName(user.GetUsername());                      //设置用户名
        zhi.SetDZ("类别删　");                                 //设置动作
        zhi.sql_insert();                                     //插入日志

}
```



添加 OnInitDialog 函数，将代码“t=CTime::GetCurrentTime;”写入函数中。



6.9　文档管理模块设计

6.9.1　文档管理模块概

文档管理模块用于查看、添加、修改和删除文档信息。“文档管理”对话框的运行效果如图 6.19 所示。



图 6.19　“文档管理”对话框的运行效果





6.9.2　文档管理模块技术分析

在文档管理模块中会应用到 InsertColumn 函数，其作用是向列表控件中插入列，语法如下：

```
InsertColumn ([位置 i],[列名 s],[对　方式 s],[宽度 i],[索引 i])
```

☑　位置：在列表中要插入的列所在的位置。
☑　列名：在列表中显示的列标题。
☑　对齐方式：在列表中选择项目的对齐方式。
☑　宽度：在列表中设置列的宽度。
☑　索引：在列表中设置列的索引号。

6.9.3　文档管理模块实现　程

文档管理模块实现过程如下：

（1）在工程中新建一个对话框，将对话框的 ID 设为 IDD_WDgldlg。

（2）向对话框中添加控件资源。各控件的摆放位置如图 6.20 所示。

（3）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.21 所示。

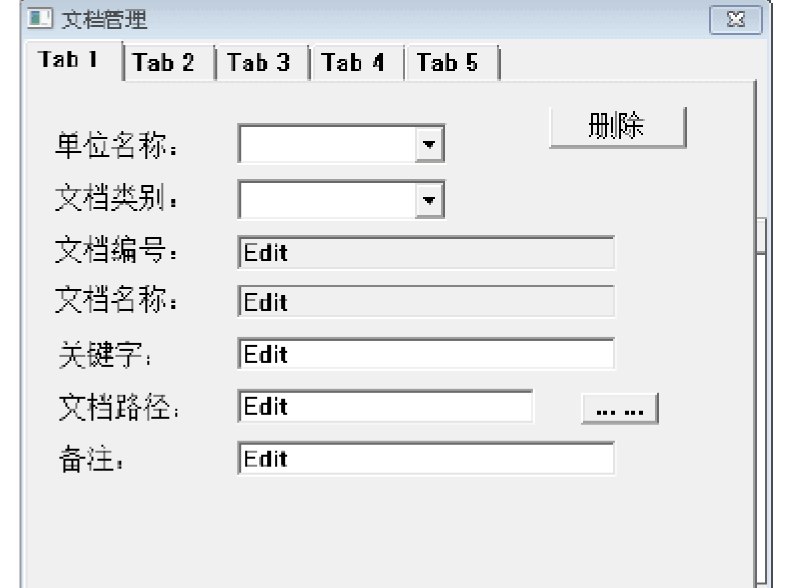


图 6.20　“文档管理”对话框

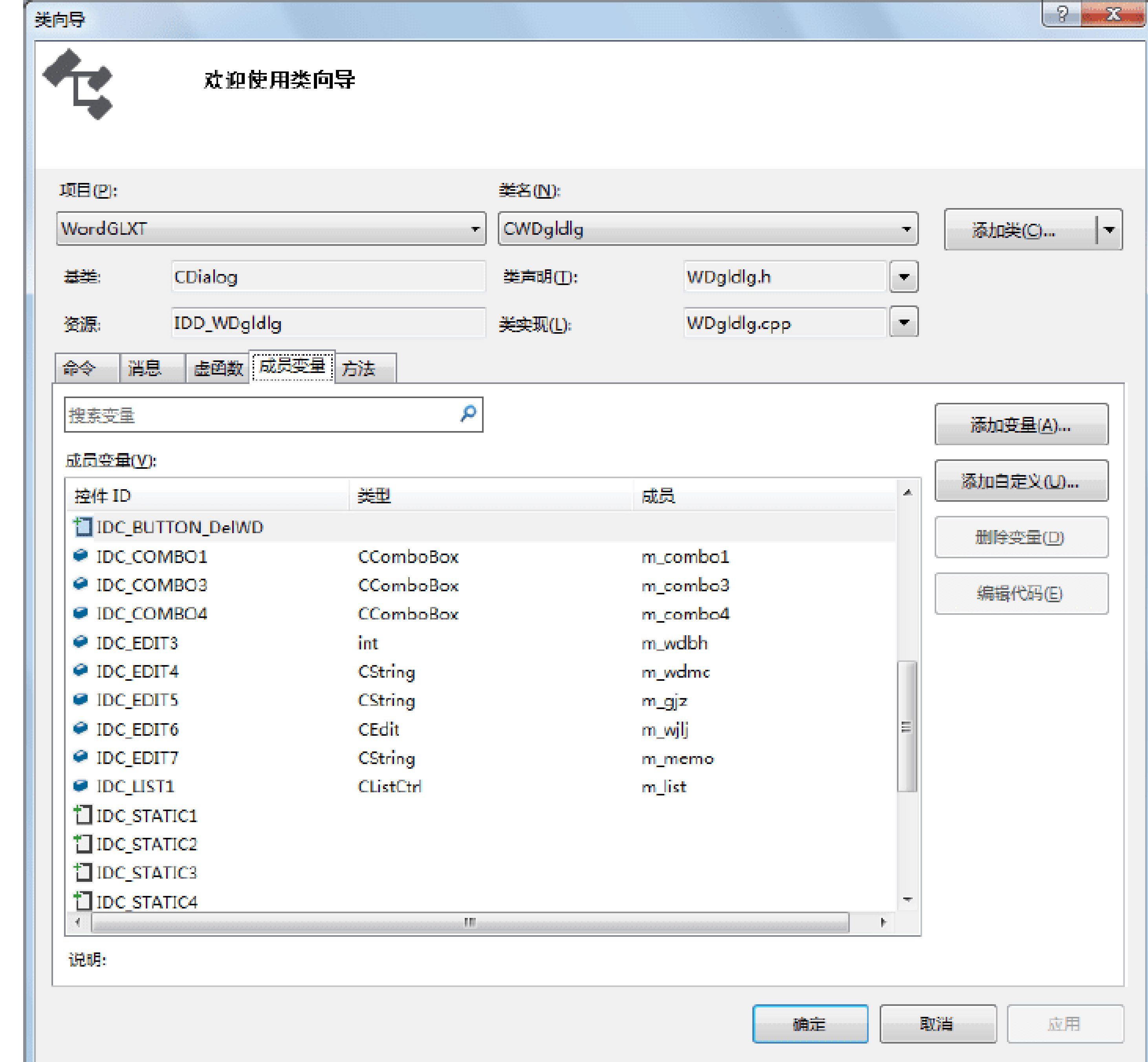


图 6.21　“类向导”对话框



（4）使用函数外部的变量的代码如下：

```
extern CUsers user;
```

在头文件中定义变量的代码如下：

```
int wdbh;                                               //文档编号
int lbbh;                                               //文档类别编号
int dwbh;                                               //单位名称编号
int str;                                                //   中按
CString strText;
CDwxxb dwb;
CZdmlb mlb;
CZdxxb xxb;
CRizhib zhi;
CTime t;
UINT tabindex;
```

添加 OnInitDialog 函数，此函数用于初始化 Tab 控件以及为 ListControl 控件赋值，代码如下：

```
BOOL CWDgldlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_hIcon = AfxGetApp()->LoadIcon(IDI_CHILDICON4);
    SetIcon(m_hIcon, TRUE);
    TC_ITEM tci;
    tci.mask=TCIF_TEXT;
    tci.pszText="基本信息";
    m_tab.InsertItem(0,&tci);
    tci.pszText="信息删   ";
    m_tab.InsertItem(1,&tci);
    dwb.Load_dep();
    mlb.Load_dep();
    xxb.Load_dep();
    t=CTime::GetCurrentTime();
    UpdateData(true);
     //根据文档编号在文档表中搜索文档名称
    for(int i=0;i<xxb.a_WDbh.GetSize();i++)
     //往      卡 2 的下拉列表框添加文档名称
        m_combo1.AddString(xxb.a_WDmc.GetAt(i));
     //根据单位编号在单位表中搜索单位名称
    for(int i=0;i<dwb.a_DWbh.GetSize();i++)
     //往      卡 1 的下拉列表框添加单位名称
        m_combo3.AddString(dwb.a_DWmc.GetAt(i));
❶         m_list.SetExtendedStyle(LVS_EX_FLATSB|LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
    m_list.InsertColumn(0,"单位名称",LVCFMT_LEFT,100,0);
    m_list.InsertColumn(1,"文档类别",LVCFMT_LEFT,100,1);
    m_list.InsertColumn(2,"文档编号",LVCFMT_LEFT,100,2);
    m_list.InsertColumn(3,"文档名称",LVCFMT_LEFT,100,3);
    m_list.InsertColumn(4,"关    字",LVCFMT_LEFT,100,4);
```

```
    m_list.InsertColumn(5,"文档    径",LVCFMT_LEFT,100,5);
    m_list.InsertColumn(6,"备注",LVCFMT_LEFT,100,6);
    CString dwmc[100],wdlb[100],pp;
//根据单位编号    回单位名称
int i=0;
    for(i=0;i<xxb.a_WDmc.GetSize();i++)
    {
        for(int j=0;j<dwb.a_DWbh.GetSize();j++)
        {
            if(atoi(xxb.a_DWbh.GetAt(i))==atoi(dwb.a_DWbh.GetAt(j)))
            {
                dwmc[i]= dwb.a_DWmc.GetAt(j);
            }
        }
        //根据类别编号    回类别名称
        for(int j=0;j<mlb.a_DWbh.GetSize();j++)
        {
            if(atoi(xxb.a_DWbh.GetAt(i))==atoi(mlb.a_DWbh.GetAt(j)) &&
                atoi(xxb.a_LBbh.GetAt(i))==atoi(mlb.a_LBbh.GetAt(j)))
            {
                wdlb[i]= mlb.a_LBmc.GetAt(j)
            }
        }
    }
    ADOConn m_AdoConn;
    m_AdoConn.OnInitADOConn();
    CString sql;
    sql.Format("select* from Zdxxb order by wdbh desc");
    m_AdoConn.GetRecordSet((_bstr_t)sql);
    while(m_AdoConn.m_pRecordset->adoEOF==0)
    {
❷       m_list.InsertItem(0,"");
        m_list.SetItemText(0,0,dwmc[i-1]);
        m_list.SetItemText(0,1,wdlb[i-1]);
        m_list.SetItemText(0,2,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wdbh"));
        m_list.SetItemText(0,3,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wdmc"));
        m_list.SetItemText(0,4,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("gjz"));
        m_list.SetItemText(0,5,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wjlj"));
        m_list.SetItemText(0,6,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("memo"));
        i--;
        m_AdoConn.m_pRecordset->MoveNext();
    }
    m_AdoConn.ExitConnect();
```

```
    //根据菜单    使不同的单  按  处于  中状态
    if(str==0)
    {
        CButton* tempbutton = (CButton*)GetDlgItem(IDC_RADIO1);
        tempbutton->SetCheck(1);
    }
    else
    {
        CButton* tempbutton = (CButton*)GetDlgItem(IDC_RADIO2);
        tempbutton->SetCheck(1);
    }
    //调用 SetCurTab()
    SetCurTab(tabindex);
    m_wdbh = xxb.a_WDmc.GetSize()+1;                    //使  认文档编号为自动排序
    UpdateData(false);
    return TRUE;
}
```

代码贴士

❶ SetExtendedStyle 函数：设置列表框的显示风格。

❷ InsertItem 函数：可以在列表框中指定位置插入一项。

（5）为“……”按钮添加如下代码，实现查找文件路径的功能。

```
void CWDgldlg::OnWjljxz()
{
    CFileDialog file(true,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,"Al Files(*.*)|*.*| |",
AfxGetMainWnd());                                       //构  文件对话框
    if(file.DoModal()==IDOK)                            //显示文件对话框
    {
❶           strText=file.GetPathName();                 //获得文件  径
        m_wjlj.SetWindowText(strText);                  //显示文件  径
        m_wdmc = file.GetFileName();                    //自动添加文档名称
        int index=m_wdmc.ReverseFind('.');
        CString tem = m_wdmc;
        if(index!=-1) tem.Delete(index, m_wdmc.GetLength()-index);
        m_gjz   = tem;
        UpdateData(false);                              //将变  m_wdmc 的数据  出到编  框中
    }
}
```

代码贴士

❶ GetPathName 函数：取得指定文件的完整路径。

当用户单击“保存”按钮时，将执行 OnOK 函数，代码如下：

```
void CWDgldlg::OnOK()
{
    UpdateData(true);
```



```
CString strdwmc,strwdlb;
if(m_combo3.GetCurSel()==CB_ERR)
{
    MessageBox("单位名称不能为空，请　择单位!","文档管理系统");
    return;
}
else
    m_combo3.GetLBText(m_combo3.GetCurSel(),strdwmc);
if(m_combo4.GetCurSel()==CB_ERR)
{
    MessageBox("文档类别不能为空，请　择文档类别!","文档管理系统");
    return;
}
else
    m_combo4.GetLBText(m_combo4.GetCurSel(),strwdlb);
if(m_wdmc=="")
{
    MessageBox("文档名称不能为空","文档管理系统");
    return;
}
CString strwjlj;
m_wjlj.GetWindowText(strwjlj);
if(strwjlj=="")
{
    MessageBox("文档　径不能为空", "文档管理系统");
    return;
}
int dw=0,lb=0;
for(int i=0;i<mlb.a_DWbh.GetSize();i++)                    //根据单位编号搜索单位类别
{

if(strdwmc==dwb.a_DWmc.GetAt(i))
        {
            dwbh=atoi(dwb.a_DWbh.GetAt(i));
            dw++;
        }
 }
 if(dw==0)
 {
     MessageBox("单位名称不存在","文档管理系统");
     return;
 }
 for(int i=0;i<mlb.a_DWbh.GetSize();i++)                   //根据单位编号搜索单位类别
 {
     if(dwbh==atoi(mlb.a_DWbh.GetAt(i)) && strwdlb==mlb.a_LBmc.GetAt(i))
     {
         lbbh=atoi(mlb.a_LBbh.GetAt(i));                    //类别编号
         lb++;
     }
```

```
    }
    if(lb==0)
    {
        MessageBox("文档类别不存在","文档管理系统");
        return;
    }
    //设置文档信息
    xxb.SetDWbh(dwbh);
    xxb.SetLBbh(lbbh);
    xxb.SetWDbh(m_wdbh);
    xxb.SetWDmc(m_wdmc);
    xxb.SetGJz(m_gjz);
    xxb.SetWJlj(strwjlj);
    xxb.SetMemo(m_memo);
    xxb.SetTjrxm(user.GetUsername());
    switch(str)
    {
    case 0:                                                  //添加
        if(xxb.HaveId(dwbh,lbbh,m_wdbh)==1)
        {
            MessageBox("文档已存在","文档管理系统");
            return;
        }
        xxb.sql_insert();                                    //插入语句
        zhi.SetDLsj(t.Format("%y-%m-%d"));                   //设置登录时
        zhi.SetName(user.GetUsername());                     //设置用户名
        zhi.SetDZ("添加文档");                               //设置动作
        zhi.sql_insert();                                    //插入日志
        break;
    case 1:                                                  //修改
        xxb.sql_update(m_wdbh);                              //修改语句
        zhi.SetDLsj(t.Format("%y-%m-%d"));                   //设置登录时
        zhi.SetName(user.GetUsername());                     //设置用户名
        zhi.SetDZ("修改文档");                               //设置动作
        zhi.sql_insert();                                    //插入日志
        break;
    }
    dw=0;
    lb=0;
    CDialog::OnOK();
}
```

为 Radio 控件添加消息响应函数，代码如下：

```
void CWDgldlg::OnRadio1()
{
    str=0;
}
void CWDgldlg::OnRadio2()
```



```
{
    str=1;
}
```

当用户单击“删除”按钮时，将执行 OnBUTTONDelWD 函数，代码如下：

```
void CWDgldlg::OnBUTTONDelWD() //删    文档的按    响应函数
{

    CString wdmc;

    if(m_combo1.GetCurSel()==CB_ERR)
    {
        MessageBox("文档名称不能为空,请    择文档!","文档管理系统");
        return;
    }
    else
        m_combo1.GetLBText(m_combo1.GetCurSel(),wdmc);

    for(int i=0;i<xxb.a_WDbh.GetSize();i++)
    {
        if(wdmc==xxb.a_WDmc.GetAt(i))
        {
            wdbh=atoi(xxb.a_WDbh.GetAt(i));
        }
    }
    xxb.sql_delete(wdbh);
    zhi.SetDLsj(t.Format("%y-%m-%d"));
    zhi.SetName(user.GetUsername());
    zhi.SetDZ("文档删    ");
    zhi.sql_insert();
    CDialog::OnOK();

}
```

实现在“单位名称”下拉列表框中选择单位时，自动在“文档类别”下拉列表框中显示与该单位对应的文档类别，代码如下：

```
//    择单位时，自动在“文档类别”下拉列表框（Combo4）中添加文档类别
void CWDgldlg::OnSelchangeCombo3()
{
    UpdateData(TRUE);
    CString strdwmc;
    //获得当前    中的单位名称
    m_combo3.GetLBText(m_combo3.GetCurSel(),strdwmc);
    dwb.Load_dep();
    mlb.Load_dep();
    xxb.Load_dep();
```

```
	m_combo4.ResetContent();                                    //删    数据
	//根据单位编号在单位表中搜索单位名称
	for(int i=0;i<dwb.a_DWbh.GetSize();i++)
	{
		if(strdwmc == dwb.a_DWmc.GetAt(i))
		{
			//根据类别编号在类别表中搜索类别名称
			for(int j=0;j<mlb.a_LBbh.GetSize();j++)
				if(atoi(dwb.a_DWbh.GetAt(i))==atoi(mlb.a_DWbh.GetAt(j)))
					//往标签 1 的“文档类别”下拉列表框添加单位名称
					m_combo4.AddString(mlb.a_LBmc.GetAt(j));
		}
	}
}
```

通过 SetCurTab 函数，根据菜单的消息响应确定显示 Tab 标签控件的第几页，代码如下：

```
void CWDgldlg::SetCurTab(UINT m_index)
{
	m_tab.SetCurSel(m_index);
	if(m_index==0)
	{
		//标签 1 的控件    藏
		GetDlgItem(IDC_LIST1)->ShowWindow(SW_HIDE);
		GetDlgItem(IDC_COMBO1)->ShowWindow(SW_HIDE);
		GetDlgItem(IDC_BUTTONDEL)->ShowWindow(SW_HIDE);
		//标签 0 的控件显示
		GetDlgItem(IDC_COMBO3)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_COMBO4)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_EDIT3)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_EDIT4)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_EDIT5)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_EDIT6)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_EDIT7)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC1)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC2)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC3)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC4)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC5)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC7)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC7)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_STATIC8)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_WJLJXZ)->ShowWindow(SW_SHOW);
		GetDlgItem(IDOK)->ShowWindow(SW_SHOW);
		GetDlgItem(IDCANCEL)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_RADIO1)->ShowWindow(SW_SHOW);
		GetDlgItem(IDC_RADIO2)->ShowWindow(SW_SHOW);
	}
	else
```



```
    {   //标签 0 的控件  藏
        GetDlgItem(IDC_COMBO3)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_COMBO4)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT3)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT4)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT5)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT6)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_EDIT7)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC1)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC2)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC3)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC4)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC5)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC6)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC7)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_STATIC8)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_WJLJXZ)->ShowWindow(SW_HIDE);
        GetDlgItem(IDOK)->ShowWindow(SW_HIDE);
        GetDlgItem(IDCANCEL)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_RADIO1)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_RADIO2)->ShowWindow(SW_HIDE);
        //标签 1 的控件显示
        GetDlgItem(IDC_LIST1)->ShowWindow(SW_SHOW);
        GetDlgItem(IDC_COMBO1)->ShowWindow(SW_SHOW);
        GetDlgItem(IDC_BUTTONDEL)->ShowWindow(SW_SHOW);
    }
}
```

6.10　口令修改模块设计



6.10.1　口令修改模块概

口令修改模块用于修改用户口令，“口令修改”对话框的运行效果如图 6.22 所示。



图 6.22　“口令修改”对话框的运行效果

6.10.2　口令修改模块实现　程

口令修改模块实现过程如下：

（1）在工程中新建一个对话框，将对话框的 ID 设为 IDD_KLXG。

（2）向对话框中添加控件资源。各控件的摆放位置如图 6.23 所示。

（3）按 Shift+Ctrl+X 组合键，打开“类向导”对话框，选择“成员变量”选项卡，为控件设置变量，如图 6.24 所示。





图 6.23 “口令修改”对话框

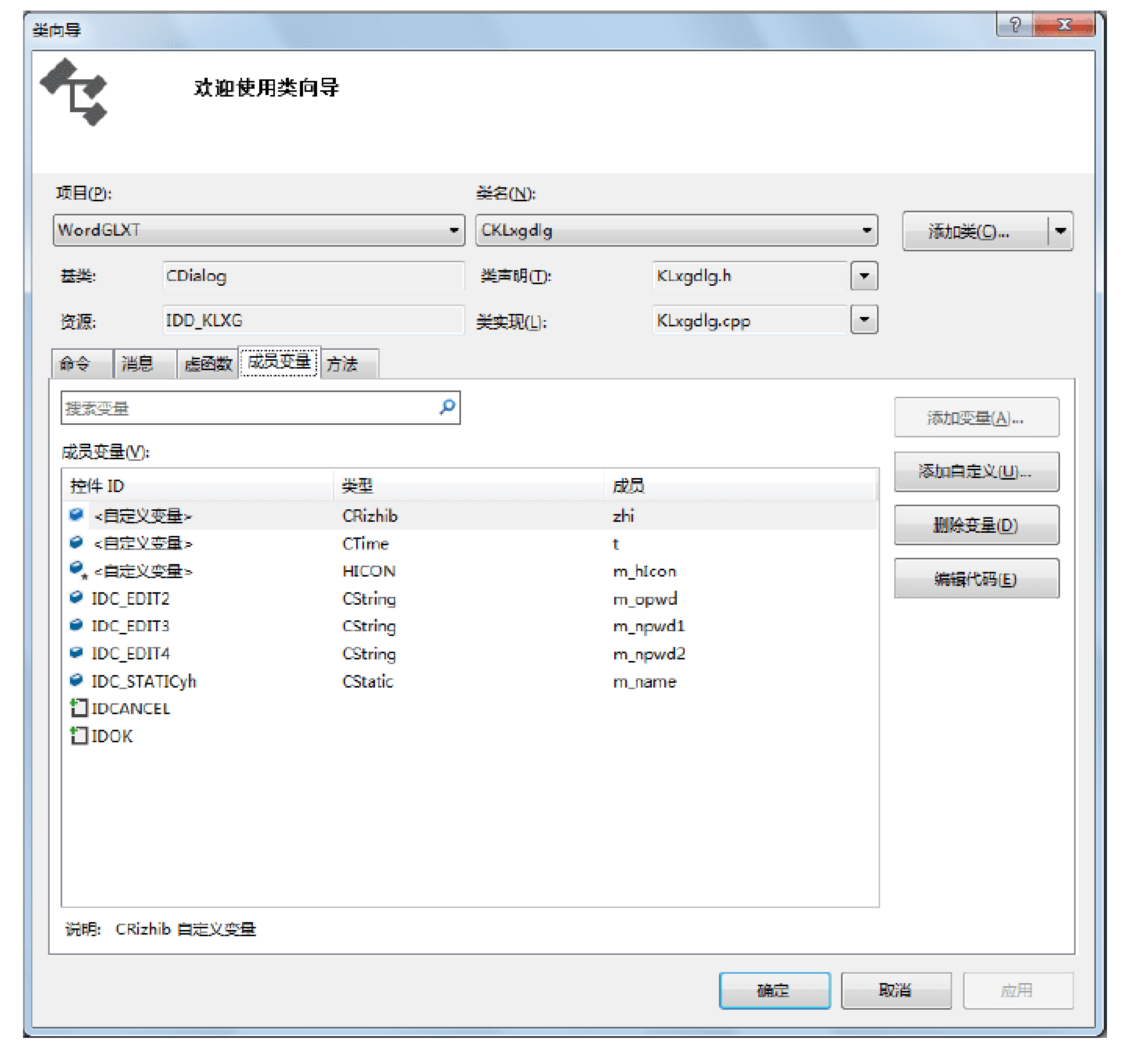


图 6.24 “类向导”对话框

（4）使用函数外部的变量的代码如下：

```
extern CUsers user;
```

在头文件中定义变量的代码如下：

```
CRizhib zhi;
CTime t;
```

添加 OnInitDialog 函数使用户名显示在文本框中，代码如下：

```
BOOL CKLxgdlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(m_hIcon, TRUE);
    t=CTime::GetCurrentTime();                          //获得当前时
    m_name.SetWindowText(user.GetUsername());           //显示登录用户
    UpdateData(false);
    return TRUE;
}
```

为“确认”按钮添加单击事件，代码如下：

```
void CKLxgdlg::OnOK()
{
```

```
    UpdateData(true);
    CString name;
    m_name.GetWindowText(name);
    if(m_opwd=="")
    {
        MessageBox("请  入旧密码","文档管理系统");
        return;
    }
    if(m_npwd1=="")
    {
        MessageBox("请  入新密码","文档管理系统");
        return;
    }
    if(m_npwd2=="")
    {
        MessageBox("请确认新密码","文档管理系统");
        return;
    }
    if(m_npwd1!=m_npwd2)
    {
        MessageBox("两次  入密码不同","文档管理系统");
        return;
    }
    CUsers ser;
    if(ser.HaveCzy(name,m_opwd)!=1)
    {
        MessageBox("用户或密码  误","文档管理系统");
        return;
    }
    else
    {
        ser.SetPwd(m_npwd1);                                    //设置用户密码
        ser.sql_updatepwd(name);                                //修改用户密码
        MessageBox("密码修改成功，下次登录请用新密码");
    }
    zhi.SetDLsj(t.Format("%y-%m-%d"));                          //设置登录时
    zhi.SetName(user.GetUsername());                            //设置用户名
    zhi.SetDZ("修改密码");                                        //设置动作
    zhi.sql_insert();                                           //插入日志
    UpdateData(false);
    CDialog::OnOK();
}
```

6.11　开发问题解析

6.11.1　怎样将数据表中的数据添加到 ListControl 控件中

在文档管理系统的多个模块中用 ListControl 控件来实现对数据表的显示，如图 6.25 所示。

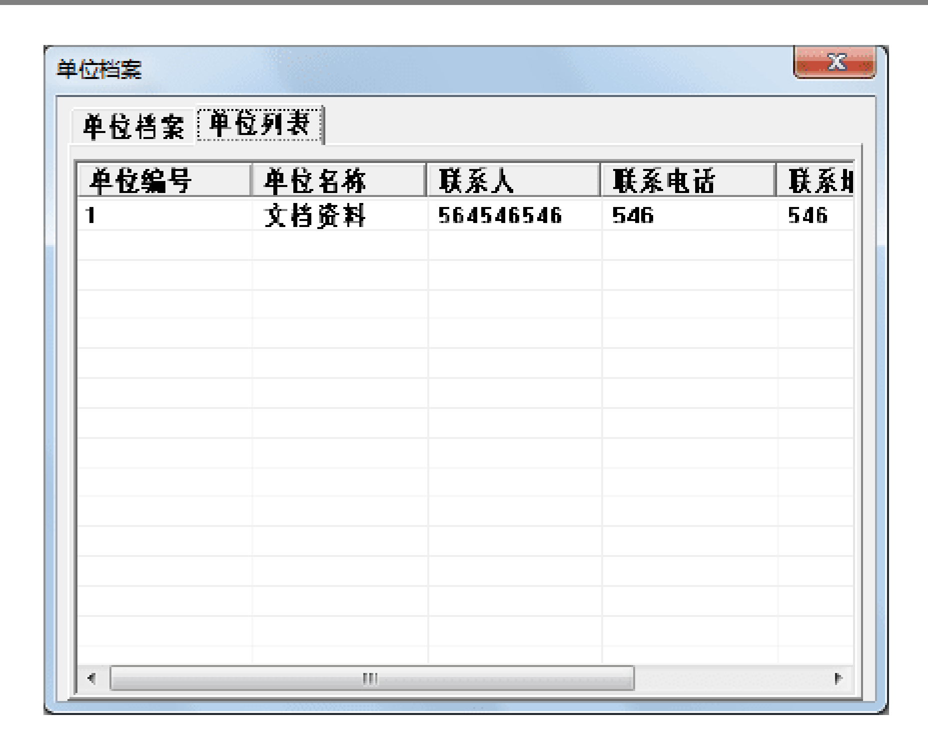


图 6.25　单位列表

从图 6.25 中可以看到，在列表中显示了需要的字段信息，ListControl 提供了成员函数 SetItemText，用它可以将某字符串添加到列表中，只要从第 1 条记录开始循环，将每条记录中需要的字段信息取出，再用 SetItemText 添加到列表中即可，代码如下：

```
...
 ADOConn m_AdoConn;
 m_AdoConn.OnInitADOConn();                                    //   接数据库
 CString sql;
 sql.Format("select* from Zdxxb order by wdbh desc");          //设置查询语句
 m_AdoConn.GetRecordSet((_bstr_t)sql);                         //   行查询
 while(m_AdoConn.m_pRecordset->adoEOF==0)
 {
      m_list.InsertItem(0,"");                                 //插入行
      m_list.SetItemText(0,0,dwmc[i-1]);
     m_list.SetItemText(0,1,wdlb[i-1]);
//插入列信息
      m_list.SetItemText(0,2,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wdbh"));
      m_list.SetItemText(0,3,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wdmc"));
      m_list.SetItemText(0,4,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("gjz"));
      m_list.SetItemText(0,5,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("wjli"));
      m_list.SetItemText(0,6,(char*)(_bstr_t)m_AdoConn.m_pRecordset->
GetCollect("memo"));

i--;
      m_AdoConn.m_pRecordset->MoveNext();                      //移动到下一条记录
 }
 m_AdoConn.ExitConnect();                                      //关   数据库   接
 ...
```


注意 这里是以倒序方式查询的，因为编号是整型，如果以正序查询，添加到列表控件后就会倒序显示。

6.11.2　怎样取得文件的完整　径

当用户选择文件进行打开或保存操作时，要用到文件打开或保存对话框。MFC 的类 CFileDialog 可以实现这种功能。使用 CFileDialog 声明一个对象时，第一个 Bool 型参数用于指定文件的打开或保存，当为 True 时将构造一个文件打开对话框，当为 False 时将构造一个文件保存对话框。关键代码如下：

```
CFileDialog   file(true,NULL,NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,"All   Files(*.*)|*.*|   |",
AfxGetMainWnd());
      if(file.DoModal()==IDOK)                                    //若打开文件成功
      {
           strText=file.GetPathName();                            //得到文件完整　径名
m_wjlj.SetWindowText(strText);
           m_wdmc = file.GetFileName();                           //自动添加文档名称
           int index=m_wdmc.ReverseFind('.');
           CString tem = m_wdmc;
           if(index!=-1) tem.Delete(index,m_wdmc.GetLength()-index);
           m_gjz   = tem;
           UpdateData(false);                                     //将变　m_wdmc 的数据　出到编　框中
      }
```

6.12　项目文件清单

文档管理系统的项目文件清单如表 6.6 所示。

表 6.6　文档管理系统文件清单

文　件　名	文 件 类 型	说　　明	文　件　名	文 件 类 型	说　　明
WordGLXT.h	头文件	工程头文件	Rizhib.h	头文件	日志表头文件
WordGLXTDlg.h	头文件	主窗口头文件	Users.h	头文件	用户表头文件
ADOConn.h	头文件	数据库连接类	WDgldlg.h	头文件	文档管理窗口
Dwdandlg.h	头文件	单位档案窗口	Wdlbiedlg.h	头文件	文档类别
Dialogin.h	头文件	登录框头文件	YHgldlg.h	头文件	用户管理窗口
Dwxxb.h	头文件	单位表头文件	Zdmlb.h	头文件	类别表头文件
KLxgdlg.h	头文件	口令修改头文件	Zdxxb.h	头文件	文档表头文件
msword.h	头文件	引用 Word 类			

6.13　本 章 总 结

本章主要讲述了文档管理系统的开发过程，通过本章的学习，读者可以了解应用程序开发的全过程。本章的文档管理系统可以实现文件的制作、修改、传递、签订、保存、销毁和存档等一系统操作。通过本章的学习，读者可以熟练地掌握 ADO 对象连接数据库和利用 CFileStatus 类获得文档属性等知识。

第 7 章

FTP 管理系统

（Visual Studio 2017+TCP/IP 实现）

FTP 协议是 Internet 上传输文件的通用协议，该协议位于 TCP/IP 协议的应用层，因此 FTP 属于应用层的一个协议，使用也比较简单。利用 FTP 协议，可以将一个完整的文件从一个系统复制到另一个系统，但是在使用 FTP 传输文件前，需要登录 FTP 服务器，用户可以通过注册的用户名和密码登录 FTP 服务器，如果 FTP 服务器允许，用户也可以匿名登录 FTP 服务器。在本章中，笔者设计了一个 FTP 客户端软件，主要功能是实现本地系统与远程 FTP 服务器间的文件上传和下载。

通过学习本章，读者可以学到：

- 登录 FTP 服务器
- 遍历 FTP 服务器目录
- 上传文件到 FTP 服务器
- 从 FTP 服务器下载文件到本地系统
- 利用多线程实现文件的上传和下载
- 利用列表视图控件显示本地和 FTP 服务器文件
- 分割视图窗口



配置说明

7.1 开发背景

FTP 文件传输协议是 Internet 上最早出现的，同时也是应用最广泛的，直到今天它仍是最重要和最基础的应用之一。FTP 提供了交互式访问，允许客户指明文件类型和格式；同时 FTP 屏蔽了各种计算机系统的细节，因而适合在局域网中任意计算机之间传输文件。由于 FTP 操作简单，开放性强，且能充分利用 Internet 来进行信息传递交流，所以目前越来越多的 FTP 服务器连入 Internet，这样越来越多的资源可以通过匿名的 FTP 来获得。据统计，全世界现在已经有数千个 FTP 文件服务器对所有的 Internet 用户开放使用，用户可以通过与 Internet 相连到远程计算机，把自己需要的文件传输过来或者把自己收集的文件上传以与他人共享。

7.2 需求分析

目前，使用 FTP 传输文件的用户越来越多，在局域网内任意计算机之间可以共享资源，用户可以通过 Internet 连接远程计算机，实现上传或下载，通过对数据统计的研究和分析，要求本系统应该具有以下功能。

☑ FTP 文件的多任务上传。
☑ FTP 文件的多任务下载。

7.3 系统设计

7.3.1 系统目标

对于 FTP 管理系统，要满足使用方便和操作灵活等设计需求。本系统应该实现以下几个功能。

☑ FTP 文件交互方式。
☑ 能够浏览磁盘文件。
☑ 系统运行稳定、安全可靠。

7.3.2 系统功能结构

系统功能结构如图 7.1 所示。

7.3.3 系统预览

FTP 文件传输管理软件只包含一个主对话框，但是主对话框却由登录信息栏、工具栏、本地信息窗口、远程 FTP 服务器信息窗口和任务列表共 5 个子窗口构成。下面分别给出各个窗口的效果图。

FTP 文件传输管理软件主窗口效果如图 7.2 所示。

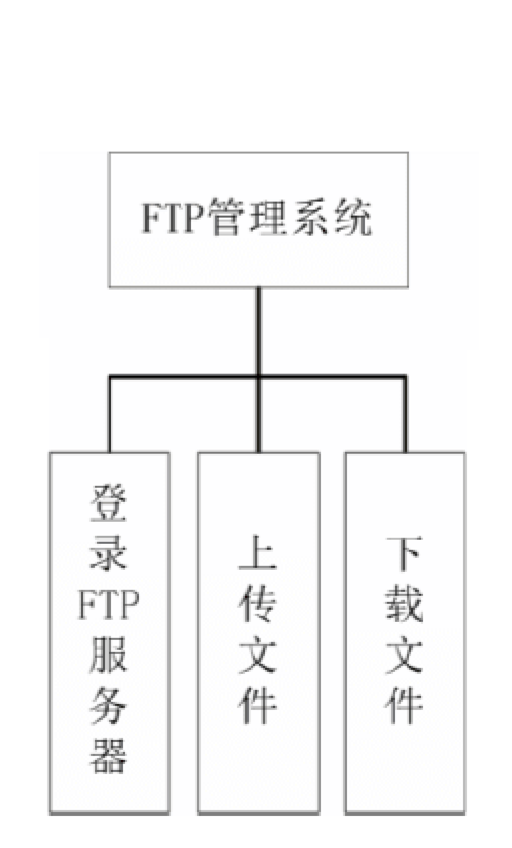


图 7.1　FTP 管理系统功能图

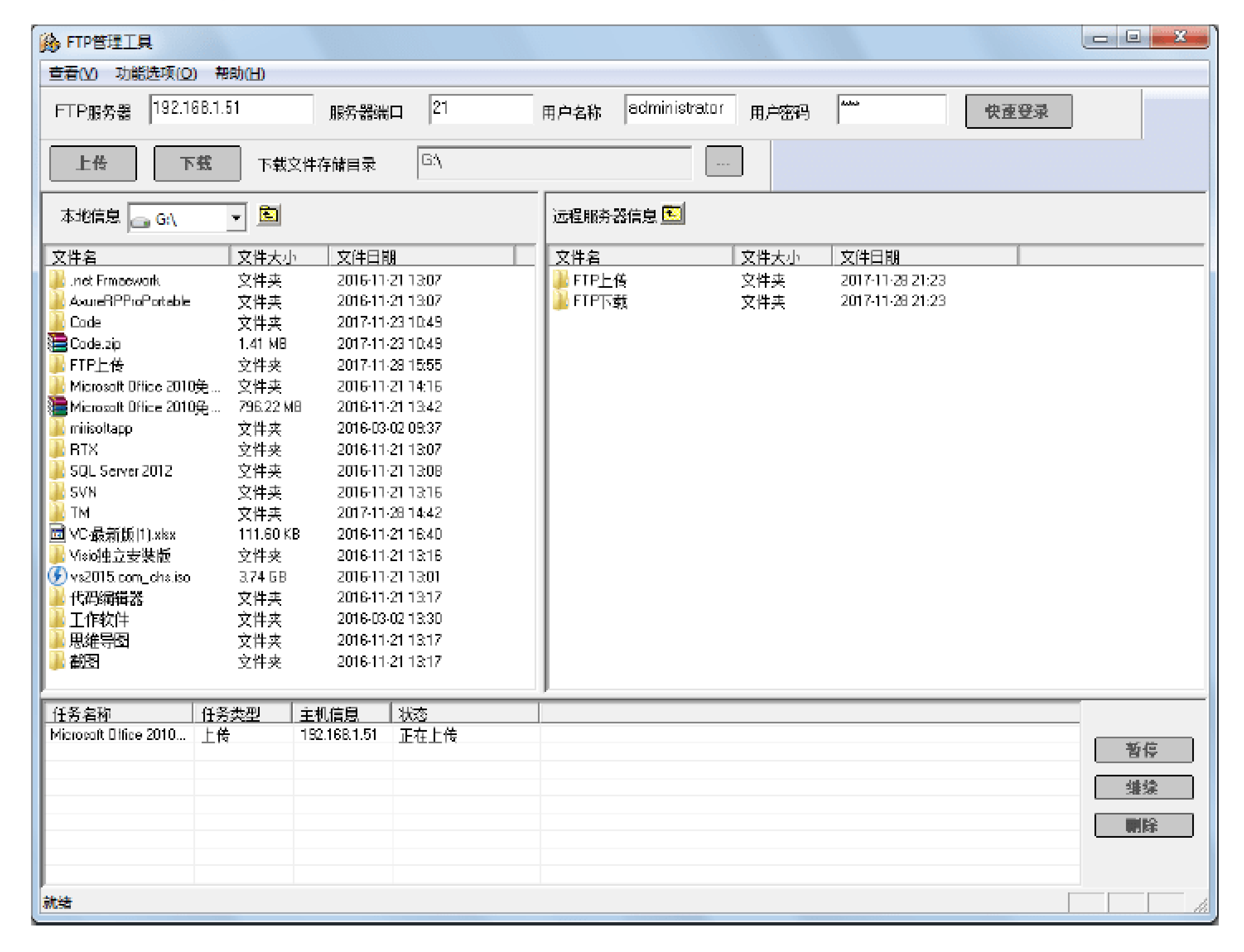


图 7.2　FTP 文件传输管理软件主窗口

登录信息栏效果如图 7.3 所示。

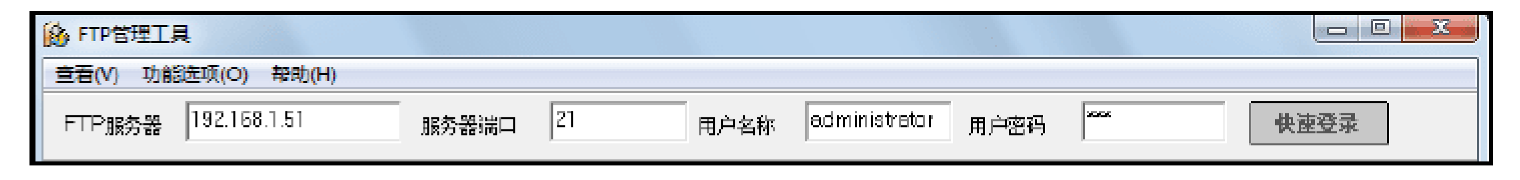


图 7.3　登录信息栏窗口

工具栏窗口效果如图 7.4 所示。

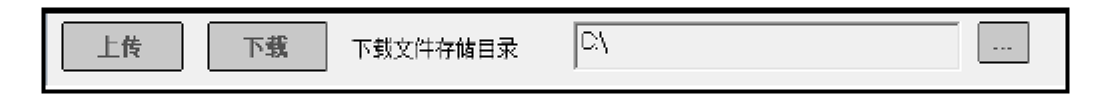


图 7.4　工具栏窗口

本地信息窗口效果如图 7.5 所示。

远程 FTP 服务器信息窗口效果如图 7.6 所示。

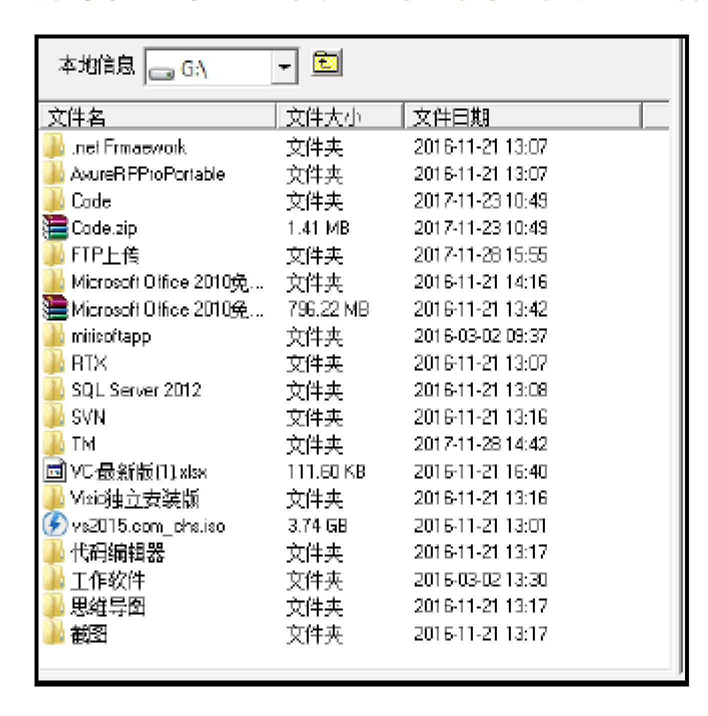


图 7.5　本地信息窗口

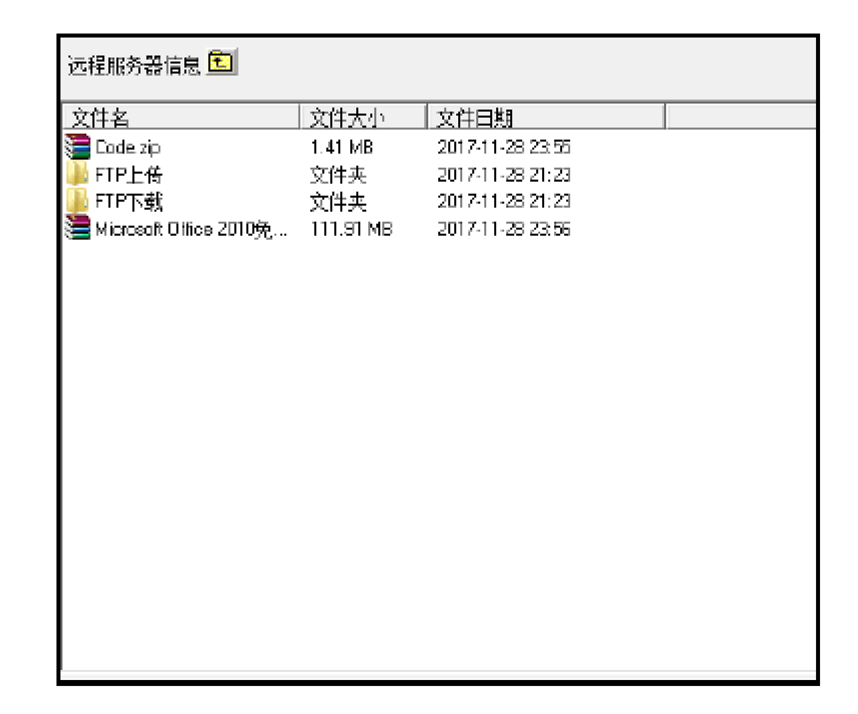


图 7.6　远程 FTP 服务器信息窗口





任务列表窗口运行效果如图 7.7 所示。

任务名称	任务类型	主机信息	状态
Microsoft Office 2010...	上传	192.168.1.51	正在上传

暂停　继续　删除

图 7.7　任务列表窗口

7.3.4　业务流程图

FTP 管理系统的业务流程图如图 7.8 所示。

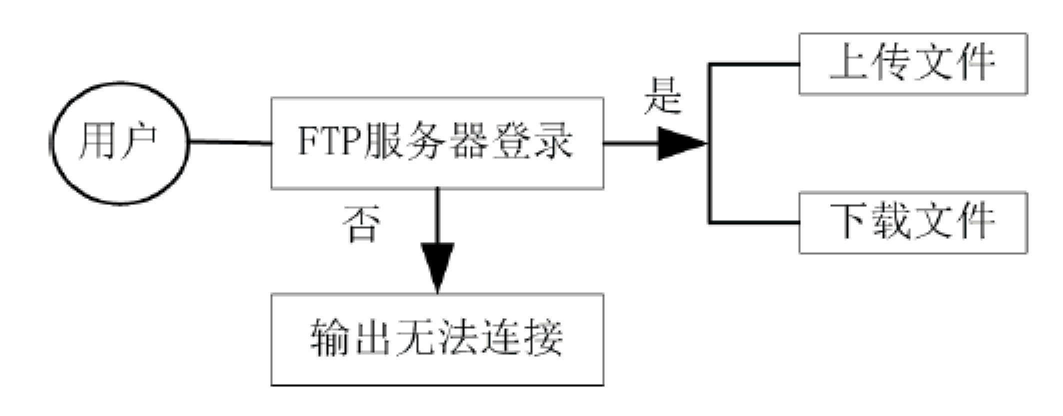


图 7.8　FTP 管理系统业务流程图

7.4　关键技术分析



7.4.1　设计类似于资源管理器的列表视图控件

在设计 FTP 文件传输管理软件时，首先需要确定采用何种方式显示本地和 FTP 服务器上的目录和文件。为了模仿 Windows 资源管理器的效果，笔者采用了列表视图控件——CListCtrl 来实现目录和文件的显示。但是，MFC 提供的默认的 CListCtrl 无法实现 Windows 资源管理器的效果，必须重新设计一个列表视图控件。该控件需要具备的功能有以本地系统默认的图标显示目录和文件的图标，在控件中双击某一个目录将进入子目录，按 Backspace 键将返回上一级目录，实现对某一列的升序、降序排列，并以箭头标识。在设计控件之前，读者需要对 CListCtrl 控件有所了解。CListCtrl 控件主要由两部分构成，第一部分是列头部分，由 CHeaderCtrl 控件构成，第二部分是表格部分。当在列头部分绘制排序箭头时，实际上是在 CHeaderCtrl 控件上进行的。

1．设置列头控件

下面介绍控件的详细设计过程。首先设计列头控件，因为需要绘制排序列的标记。

（1）从 CHeaderCtrl 类派生一个子类——CSortHeaderCtrl，向该类中添加成员变量。代码如下：

```
Int m_nSortColumn;                              //排序列
BOOL m_bAscend;                                 //是否为升序
```

（2）在构造函数中初始化成员变量。代码如下：



```
CSortHeaderCtrl::CSortHeaderCtrl()
{
	m_nSortColumn = -1;
	m_bAscend = TRUE;
}
```

（3）向 CSortHeaderCtrl 类中添加 SetSortColomn 方法，用于设置排序列和排序列的自绘风格。代码如下：

```
void CSortHeaderCtrl::SetSortColomn(int nColumn, BOOL bAscend)
{
	m_nSortColumn = nColumn;					//设置排序列
	m_bAscend = bAscend;						//设置排序方式
	HD_ITEM hItem;
	hItem.mask = HDI_FORMAT;
	GetItem(nColumn, &hItem);					//获取列信息
	hItem.fmt |= HDF_OWNERDRAW;				//设置列自绘风格
	SetItem(nColumn, &hItem);					//设置列信息
	Invalidate();								//更新控件
}
```

（4）改写 CSortHeaderCtrl 类的 DrawItem 方法，根据排序方式绘制排序列的箭头符号。代码如下：

```
void CSortHeaderCtrl::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
	CDC dc;									//定义设备上下文
	dc.Attach(lpDrawItemStruct->hDC);			//附加设备上下文句柄
	const int nSavedIndex = dc.SaveDC();		//保存设备上下文
	CRect rc(lpDrawItemStruct->rcItem);			//获取当前列区域
	CBrush brush(GetSysColor(COLOR_3DFACE));	//定义背景画刷
	dc.FillRect(rc, &brush);					//填充画刷
	TCHAR szText[256];						//定义字符数组，存储列文本
	HD_ITEM hditem;
	hditem.mask = HDI_TEXT | HDI_FORMAT;
	hditem.pszText = szText;
	hditem.cchTextMax = 255;
	GetItem(lpDrawItemStruct->itemID, &hditem);	//获取当前的项目信息
	UINT uFormat = DT_SINGLELINE | DT_NOPREFIX | DT_NOCLIP |
		DT_VCENTER | DT_END_ELLIPSIS;			//设置绘制的文本格式
	if(hditem.fmt & HDF_CENTER)
		uFormat |= DT_CENTER;
	else if(hditem.fmt & HDF_RIGHT)
		uFormat |= DT_RIGHT;
	else
		uFormat |= DT_LEFT;
	if(lpDrawItemStruct->itemState == ODS_SELECTED)	//列是否被选中
	{
		rc.left++;							//调整列的区域
		rc.top += 2;
		rc.right++;
	}
```



```
    CRect rcIcon(lpDrawItemStruct->rcItem);                        //定义箭头显示区域
    const int iOffset = (rcIcon.bottom - rcIcon.top) / 4;
    if(lpDrawItemStruct->itemID == (UINT) m_nSortColumn)
        rc.right -= 3 * iOffset;
    rc.left += iOffset;
    rc.right -= iOffset;
    if(rc.left < rc.right)
        dc.DrawText(szText, -1, rc, uFormat);                      //绘制列文本
    //绘制箭头
    if(lpDrawItemStruct->itemID == (UINT) m_nSortColumn)
    {

        CPen penLight(PS_SOLID, 1, GetSysColor( COLOR_3DHILIGHT));        //定义浅颜色画笔
        CPen penShadow(PS_SOLID, 1, GetSysColor( COLOR_3DSHADOW));    //定义深颜色画笔
        CPen* pOldPen = dc.SelectObject(&penLight);                //选中画笔
        if(m_bAscend)                                              //绘制向上的箭头
        {
            dc.MoveTo(rcIcon.right - 2 * iOffset, iOffset);
            dc.LineTo(rcIcon.right - iOffset, rcIcon.bottom - iOffset - 1);
            dc.LineTo(rcIcon.right - 3 * iOffset - 2, rcIcon.bottom - iOffset - 1);
            dc.SelectObject(&penShadow);
            dc.MoveTo(rcIcon.right - 3 * iOffset - 1, rcIcon.bottom - iOffset - 1);
            dc.LineTo(rcIcon.right - 2 * iOffset, iOffset - 1);
        }
        else                                                       //绘制向下的箭头
        {
            dc.MoveTo(rcIcon.right - iOffset - 1, iOffset);
            dc.LineTo(rcIcon.right - 2 * iOffset - 1, rcIcon.bottom - iOffset);
            dc.SelectObject(&penShadow);
            dc.MoveTo(rcIcon.right - 2 * iOffset - 2, rcIcon.bottom - iOffset);
            dc.LineTo(rcIcon.right - 3 * iOffset - 1, iOffset);
            dc.LineTo(rcIcon.right - iOffset - 1, iOffset);
        }
        dc.SelectObject(pOldPen);                                  //恢复原来选择的画笔
    }
    dc.RestoreDC(nSavedIndex);                                     //恢复之前的设备上下文
    dc.Detach();                                                   //从设备上下文中分离设备上下文句柄
}
```

2．设置列表视图控件

在设计完列头控件之后，下面开始设计列表视图控件。

（1）从 CListCtrl 类派生一个子类——CSortListCtrl，向该类中添加如下成员变量。代码如下：

```
CString m_BaseDir;                                                 //基目录
CString m_CurDir;                                                  //记录当前列表中文件的目录
int m_nListType;                        //列表类型，0 表示显示本地信息，1 表示显示 FTP 服务器信息，默认为 0
CInternetSession m_Session;                                        //Internet 会话
CString m_FtpServer,m_Port,m_User,m_Password;                      //FTP 服务器，端口号，用户名和密码
```

```
CSortHeaderCtrl m_ctlHeader;                                    //列头
int m_nNumColumns;                                              //列数
int m_nSortColumn;                                              //排序列
BOOL m_bAscend;                                                 //是否升序排列
```

（2）定义一个类 CItemData，描述项目的额外数据，主要用于记录某一行各个列的文本。当排序时，需要根据排序函数的参数获取排序列的文本以便进行比较。代码如下：

```
class CItemData
{
    public:
    CItemData()                                                 //构造函数
    {
        m_ColumnTexts = NULL;
        m_dwData = 0;
    }
    LPTSTR* m_ColumnTexts;                                      //记录当前行所有列文本
    DWORD   m_dwData;                                           //视图项数据
    private:
    //禁止复制
    CItemData(const CItemData&);
    CItemData& operator=(const CItemData&);
};
```

（3）向 CSortListCtrl 类中添加 SetItemDataList 方法，为项目关联一个自定义的数据结构——CItemData。在进行排序时需要根据该数据结构获取排序列的文本。代码如下：

```
BOOL CSortListCtrl::SetItemDataList(int iItem, LPTSTR* pchTexts)
{
    if (CListCtrl::GetItemData(iItem) == NULL)                  //判断关联的数据是否为空
    {
        CItemData* pItemData = new CItemData();                 //构建一个 CItemData 对象
        pItemData->m_ColumnTexts = pchTexts;                    //设置列文本
        //设置项目数据
        return CListCtrl::SetItemData(iItem, (DWORD)pItemData);
    }
}
```

（4）向 CSortListCtrl 类中添加 GetItemDataList 方法，获取项目关联的数据。代码如下：

```
LPTSTR* CSortListCtrl::GetItemDataList(int iItem) const
{
    ASSERT(iItem < GetItemCount());
    CItemData* pItemData = (CItemData* )CListCtrl::GetItemData( iItem);     //获取关联数据
    return pItemData->m_ColumnTexts;                                        //返回列文本
}
```

（5）向 CSortListCtrl 类中添加 SetItemText 方法，设置项目的文本，同时修改关联的 CItemData 结构的列文本。代码如下：



```
BOOL CSortListCtrl::SetItemText(int nItem, int nSubItem, LPCTSTR lpszText)
{
	if(!CListCtrl::SetItemText(nItem, nSubItem, lpszText))	//调用基类的方法设置文本
		return FALSE;
	LPTSTR* pszTexts = GetItemDataList(nItem);	//记录各列文本
	LPTSTR pszText = pszTexts[nSubItem];	//获取当前列文本
	delete[] pszText;	//释放文本数据
	pszText = new TCHAR[lstrlen(lpszText) + 1];
	lstrcpy(pszText, lpszText);	//重新设置文本
	pszTexts[nSubItem] = pszText;
	return TRUE;
}
```

（6）向 CSortListCtrl 类中添加 AddItem 方法，用于添加新行，设置行各列文本。代码如下：

```
int CSortListCtrl::AddItem(LPCTSTR pszText, ...)
{
	int nIndex = InsertItem(GetItemCount(), pszText);	//添加行，返回行索引
	LPTSTR* pszColumnTexts = new LPTSTR[m_nNumColumns];	//记录各列文本
	pszColumnTexts[0] = new TCHAR[ lstrlen(pszText) + 1];
	lstrcpy(pszColumnTexts[0], pszText);	//设置第一列文本
	va_list list;
	va_start(list, pszText);
	for(int nColumn = 1; nColumn < m_nNumColumns; nColumn++)	//设置其他列文本
	{
		pszText = va_arg(list, LPCTSTR);
		CListCtrl::SetItem(nIndex, nColumn, LVIF_TEXT, pszText, 0, 0, 0, 0);
		pszColumnTexts[nColumn] = new TCHAR[lstrlen(pszText) + 1];
		lstrcpy(pszColumnTexts[ nColumn], pszText);
	}
	va_end(list);
	SetItemDataList(nIndex, pszColumnTexts);	//设置行关联数据
	return nIndex;
}
```

（7）向 CSortListCtrl 类中添加 SetColumns 方法，向列表视图控件中添加列，设置列文本。该方法的主要作用是可以一次添加多列，并且可以指定列的宽度。代码如下：

```
//设置列，格式为文本，宽度；文本，宽度；……
BOOL CSortListCtrl::SetColumns(const CString& strHeadings)
{
	int nStart = 0;
	for(;;)
	{
		int nComma = strHeadings.Find(_T(','), nStart);	//查找“,”标记
		if(nComma == -1)
			break;
		CString strHeading = strHeadings.Mid(nStart, nComma - nStart);	//获取文本
		nStart = nComma + 1;	//掠过“,”
```

```
            int nSemiColon = strHeadings.Find(_T(';'), nStart);           //查找“;”
            if(nSemiColon == -1)                                          //查找到了结尾
                nSemiColon = strHeadings.GetLength();
            int nWidth = atoi(strHeadings.Mid(nStart, nSemiColon - nStart)); //获取宽度
            nStart = nSemiColon + 1;                                      //指向下一列信息
            if( nsertColumn(m_nNumColumns++, strHeading, LVCFMT_LEFT, nWidth) == -1)
                return FALSE;                                             //插入列
        }
        return TRUE;
}
```

（8）向 CSortListCtrl 类中添加 GetIconFromExtendedName 方法，根据文件的类型获取系统对应的文件图标。GetIconFromExtendedName 方法首先判断参数是文件还是目录，如果是文件，则调用 SHGetFileInfo 函数并传递 SHGFI_USEFILEATTRIBUTES 标记来依据文件类型返回图标，如果为目录，则以当前目录为参数，调用 SHGetFileInfo 函数获取系统目录的图标。代码如下：

```
//根据文件类型获取图标
HICON CSortListCtrl::GetIconFromExtendedName(LPCTSTR lpName)
{
        SHFILEINFO shInfo;                                              //定义外壳文件信息
        int nIcon = 0;
        CString extension = lpName;
        CString csName = "text"+extension;                              //设置一个临时的文件名
        int nPos = csName.ReverseFind('.');                             //判断是否为文件
        if (nPos >0)
        {
            //获取文件图标
            SHGetFileInfo(csName,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
                SHGFI_ICON | SHGFI_SMALLICON|SHGFI_USEFILEATTRIBUTES);
        }
        else                                                            //参数表示一个目录
        {
            char chPath[MAX_PATH] = {0};
            GetCurrentDirectory(MAX_PATH,chPath);                       //获取当前目录
            SHGetFileInfo(chPath,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
                SHGFI_ICON | SHGFI_SMALLICON);                          //获取目录图标
        }
        nPos = shInfo.iIcon;
        return shInfo.hIcon;                                            //返回获取的图标
}
```

（9）向 CSortListCtrl 类中添加 DisplayPath 方法，列举本地或 FTP 服务器目录和文件信息。该方法非常重要，在程序中多处都需要调用 DisplayPath 方法，例如，在视图列表中显示本地系统中的某一个目录下的子目录和文件，用户按 Backspace 键返回上一级目录等，这些都需要调用 DisplayPath 方法来实现。

DisplayPath 方法代码较多，但是并不复杂。首先根据成员变量 m_nListType 来判断显示的是本地系统目录，还是 FTP 服务器上的目录。以显示本地系统目录为例，使用 CFileFind 类来遍历当前目录



的直接子目录和文件，如果是文件，则读取文件的修改日期和大小信息，如果是目录，则标记为文件夹，将这些信息添加到列表视图中，然后获取文件关联的图标索引，设置视图项显示的图像索引，最后为了区分列表视图中的项目表示的是文件还是目录，为每个视图项设置一个额外的整数值，0 表示文件，1 表示目录。代码如下：

```
void CSortListCtrl::DisplayPath(LPCTSTR lpPath,CFtpConnection* pTemp)
{
	DeleteAllItems();                                    //删除所有视图项
	if (m_nListType==0)                                  //显示本地系统信息
	{
		BOOL bFind;                                      //记录查找结果
		CFileFind flFind;                                //定义文件查找对象
		CString csPath = lpPath;                         //记录参数信息
		m_CurDir = lpPath;         //设置当前目录，进入子目录或返回上一级目录时需要依据当前目录
		if (csPath.Right(1) != "\\")                     //保证目录以“\”结尾
		{
			csPath += "\\";
			m_CurDir +="\\";
		}
		csPath += "*.*";                                 //在目录结尾添加*.*用于查找文件
		bFind = flFind.FindFile(csPath);                 //开始查找文件
		CString csText,csFileSize,csDataTime;
		while (bFind)                                    //设置列表当前显示的目录
		{
			bFind = flFind.FindNextFile();               //查找下一个文件
			if (!flFind.IsDots() && !flFind.IsHidden())
			{
				__int64 lFileLen = flFind.GetLength64();     //获取文件长度

				if (flFind.IsDirectory())                //判断是否为目录
				{
					csFileSize = "文件夹";
				}
				else                                     //如果是文件
				{
					//获取文件的大小
					double fGB = lFileLen /(double)(1024*1024*1024);
					if (fGB < 1)                         //判断文件是否小于 1GB
					{
						double fMB = lFileLen / (double)(1024*1024);
						if (fMB < 1)                     //判断文件是否小于 1MB
						{
							double fBK = lFileLen / (double)(1024);
							if (fBK >1)                  //判断文件是否小于 1KB
							{
								csFileSize.Format("%2.2f KB",fBK);
							}
							else                         //使用位作为单位
```

```
                    {
                        csFileSize.Format("%i B",lFileLen);
                    }
                }
                else
                {
                    csFileSize.Format("%2.2f MB",fMB);
                }
            }
            else
            {
                csFileSize.Format("%2.2f GB",fGB);
            }
        }
        csText = flFind.GetFileName();                    //获取文件名称
        CTime time;
        flFind.GetCreationTime(time);                     //获取文件日期
        csDataTime = time.Format("%Y-%m-%d %H:%M");  //格式化日期
        int nItem = AddItem(csText,csFileSize,csDataTime); //添加视图项
        //设置文件显示的图标
        SHFILEINFO shInfo;
        int nIcon = 0;
        SHGetFileInfo(flFind.GetFilePath(),0,&shInfo,sizeof(shInfo),SHGFI_ICON |
            SHGFI_SMALLICON);                             //获取文件图标
        DestroyIcon(shInfo.hIcon);
        nIcon = shInfo.iIcon;                             //获取文件图标索引
        SetItem(nItem,0,LVIF_IMAGE,"",nIcon,0,0,0);       //设置视图项图标
        //设置项目标记，0 表示文件，1 表示目录
        if (flFind.IsDirectory())
        {
            SetItemData(nItem,1);
        }
        else
        {
            SetItemData(nItem,0);
        }
        nItem++;
    }
}
}
else                                                      //显示 FTP 服务器信息
{
    CFtpConnection* pFTP = NULL;                          //定义 FTP 连接对象指针
    //登录 FTP 服务器
    pFTP = m_Session.GetFtpConnection(m_FtpServer,m_User,m_Password,atoi(m_Port));
    pFTP->SetCurrentDirectory("");                        //设置当前目录
    CFtpFileFind Find(pFTP);                              //定义 FTP 文件查找对象
    BOOL bFind;                                           //记录查找结果
    m_CurDir =lpPath;                                     //设置当前目录
```

```
if (strlen(lpPath)==0)
    bFind = Find.FindFile(NULL,INTERNET_FLAG_EXISTING_CONNECT|
        INTERNET_FLAG_RELOAD);                        //从 FTP 根目录开始查找
else
    bFind = Find.FindFile(lpPath,INTERNET_FLAG_EXISTING_CONNECT|
        INTERNET_FLAG_RELOAD);                        //查找指定目录
if (bFind)                                            //查找是否成功
{
    CString csFileName,csDataTime,csFileSize;
    while (bFind)                                     //遍历当前目录
    {
        bFind = Find.FindNextFile();                  //查找下一个文件
        csFileName = Find.GetFileName();              //获取文件名称
        CTime fileTime;
        Find.GetLastWriteTime(fileTime);              //获取文件修改时间
        //格式化文件修改时间
        csDataTime = fileTime.Format("%Y-%m-%d %H:%M");
        if (!Find.IsDots() && !Find.IsHidden())
        {
            __int64 lFileLen = Find.GetLength64();    //获取文件长度
            if (Find.IsDirectory())                   //判断是否为目录
            {
                csFileSize = "文件夹";
            }
            else                                      //如果是文件
            {
                double fGB = lFileLen /(double)(1024*1024*1024);
                if (fGB < 1)                          //是否小于 1GB
                {
                    double fMB = lFileLen / (double)(1024*1024);
                    if (fMB < 1)                      //是否小于 1MB
                    {
                        double fBK = lFileLen / (double)(1024);
                        if (fBK >1)                   //是否小于 1KB
                        {
                            csFileSize.Format("%2.2f KB",fBK);
                        }
                        else                          //以位为单位显示
                        {
                            csFileSize.Format("%i B",lFileLen);
                        }
                    }
                    else
                    {
                        csFileSize.Format("%2.2f MB",fMB);
                    }
                }
                else
                {
```

```
                        csFileSize.Format("%2.2f GB",fGB);
                    }
                }
            }
            int nItem = AddItem(csFileName,csFileSize,csDataTime); //添加视图项
            //设置文件显示的图标
            SHFILEINFO shInfo;                                      //定义文件外壳信息
            int nIcon = 0;
            CString csName = Find.GetFileName();                    //获取文件名
            int nPos = csName.ReverseFind('.');                     //查找扩展名
            if (nPos >0)                                            //如果是文件
            {
                //映射为本地系统的文件图标
                SHGetFileInfo(csName,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
                    SHGFI_ICON | SHGFI_SMALLICON|SHGFI_USEFILEATTRIBUTES);
            }
            else                                                    //如果是目录
            {
                char chPath[MAX_PATH] = {0};
                GetCurrentDirectory(MAX_PATH,chPath);               //获取当前目录
                SHGetFileInfo(chPath,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
                    SHGFI_ICON | SHGFI_SMALLICON);                  //获取目录图标
            }
            DestroyIcon(shInfo.hIcon);
            nIcon = shInfo.iIcon;                                   //获取图标索引
            SetItem(nItem,0,LVIF_IMAGE,"",nIcon,0,0,0);             //设置视图项显示的图标
            //设置项目标记，0 表示文件，1 表示目录
            if (Find.IsDirectory())
            {
                SetItemData(nItem,1);                               //设置视图项标记
            }
            else
            {
                SetItemData(nItem,0);                               //设置视图项标记
            }
        }
    }
    Find.Close();                                                   //关闭文件查找对象
    pFTP->Close();                                                  //关闭 FTP 文件连接
    delete pFTP;                                                    //释放 FTP 连接对象
  }
}
```

（10）处理列表视图控件的双击事件，如果用户双击的是目录项目，则进入子目录。代码首先判断当前列表视图显示的是本地系统目录信息，还是 FTP 服务器目录信息。以显示本地系统目录信息为例，首先获取当前视图项关联的整数值，来区分当前视图项表示的是文件还是目录，如果是目录，则获取该目录的完整路径，将其作为参数传递到 DisplayPath 方法中以显示直接子目录和文件。代码如下：



```
void CSortListCtrl::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
	int nItem = GetSelectionMark();					//获取当前选中的视图项索引
	if (nItem != -1)
	{
		if (m_nListType==0)					//进入本地系统子目录
		{
			int nFlag =GetItemData(nItem);			//获取视图项关联的整数值
			if (nFlag==1)					//判断是否为目录
			{
				//获取完整的目录信息
				CString csFoder = GetItemText(nItem,0);
				csFoder += "\\";
				//获取目录
				m_CurDir += csFoder;
				DisplayPath(m_CurDir);			//进入子目录
			}
		}
		else						//进入 FTP 服务器子目录
		{
			int nFlag =GetItemData(nItem);			//获取视图项关联的整数值
			if (nFlag==1)					//判断是否为目录
			{
				//获取完整的目录信息
				CString csFoder = GetItemText(nItem,0);
				csFoder += "/";
				//获取目录
				m_CurDir += csFoder;
				DisplayPath(m_CurDir);			//进入子目录
			}
		}
	}
	*pResult = 0;
}
```

（11）处理列表视图的 WM_KEYDOWN 消息，当用户按 Backspace 键时返回上一级目录。代码首先判断列表视图显示的是本地系统目录还是 FTP 服务器目录，如果是本地系统目录，则根据当前目录获取上一级目录，然后调用 DisplayPath 方法显示上一级目录。如果是 FTP 服务器目录，仍然根据当前目录获取上一级目录，只是 FTP 服务器的目录格式与本地系统的目录格式不同，最后调用 DisplayPath 方法显示上一级目录。代码如下：

```
void CSortListCtrl::OnKeydown(NMHDR* pNMHDR, LRESULT* pResult)
{
	LV_KEYDOWN* pLVKeyDow = (LV_KEYDOWN*)pNMHDR;
	if (pLVKeyDow->wVKey==VK_BACK)				//返回上一级目录
	{
		if (m_nListType == 0)				//显示本地系统目录
		{
```

```
            if (m_BaseDir != m_CurDir && m_BaseDir != "")          //如果为根目录，则取消操作
            {
                if (m_CurDir.Right(1) == "\\")
                {
                    //去除“\\”
                    m_CurDir = m_CurDir.Left(m_CurDir.GetLength()-1);
                }
                int nPos = m_CurDir.ReverseFind('\\');
                m_CurDir = m_CurDir.Left(nPos);                    //返回上级目录
            }
        }
        else                                                       //显示 FTP 服务器目录
        {
            CString csTmpDir = m_CurDir;
            CString csTmpBase = m_BaseDir;
            //去除字符串两端的“/”符号
            if (csTmpDir.Right(1)=="/")
                csTmpDir = csTmpDir.Left(csTmpDir.GetLength()-1);
            if(csTmpDir.Left(1)=="/")
                csTmpDir = csTmpDir.Mid(1);
            if (csTmpBase.Right(1)=="/")
                csTmpBase = csTmpBase.Left(csTmpBase.GetLength()-1);
            if(csTmpBase.Left(1)=="/")
                csTmpBase = csTmpBase.Mid(1);
            if (csTmpBase == csTmpDir)                             //如果当前目录已是根目录，则退出
            {
                return;
            }
            if (m_CurDir != "")
            {
                if (m_CurDir.Right(1) == "/")                      //目录是否以“/”结尾
                {
                    //去除“/”符号
                    m_CurDir = m_CurDir.Left(m_CurDir.GetLength()-1);
                }
                int nPos = m_CurDir.ReverseFind('/');              //反向查找“/”符号
                if (nPos != -1)
                    m_CurDir = m_CurDir.Left(nPos) + '/';          //获取上级目录
                else
                    m_CurDir = "";
            }
        }
        DisplayPath(m_CurDir);                                     //显示上级目录
    }
    *pResult = 0;
}
```

（12）添加全局函数 IsNumber，判断文本是否为数字。因为在排序列时，首先按数字大小排序。代码如下：



```
//判断文本是否为数据
bool IsNumber(LPCTSTR pszText)
{
	for(int i = 0; i < lstrlen(pszText); i++)			//遍历每一个字符
		if(!_istdigit(pszText[i]))				//判断是否为数值
			return false;
	return true;
}
```

（13）添加 CompareDataAsNumber 全局函数，用于进行数值比较。对于排序函数来说，如果第一项位于第二项之前，比较函数需要返回一个负数，如果第一项位于第二项之后，比较函数需要返回一个整数。如果两个项相等，比较函数需要返回零值。代码如下：

```
int CompareDataAsNumber(LPCTSTR pszParam1, LPCTSTR pszParam2)
{
	int nNumber1 = atoi(pszParam1);
	int nNumber2 = atoi(pszParam2);
	return nNumber1 - nNumber2;
}
```

（14）添加全局函数 IsDate，判断文本是否为日期。日期格式需要采用“2008-08-08”的形式。代码如下：

```
//判断是否为日期
bool IsDate(LPCTSTR pszText)
{
	//格式为 0000-00-00
	if(lstrlen(pszText) < 10)
		return false;
	return _istdigit(pszText[0])				//判断年部分是否合法
		&& _istdigit(pszText[1])
		&& _istdigit(pszText[2])
		&& _istdigit(pszText[3])
		&& pszText[4] == _T('-')
		&& _istdigit(pszText[5])				//判断月部分是否合法
		&& _istdigit(pszText[6])
		&& pszText[7] == _T('-')
		&& _istdigit(pszText[8])				//判断日部分是否合法
		&& _istdigit(pszText[9]);
}
```

（15）添加 CompareDataAsDate 全局函数，进行日期比较。日期是按年、月、日的先后顺序进行比较。代码如下：

```
//按日期比较数据
int CompareDataAsDate(const CString& strDate1, const CString& strDate2)
{
	//先比较年
```

```
    int nYear1 = atoi(strDate1.Mid(0, 4));
    int nYear2 = atoi(strDate2.Mid(0, 4));
    if (nYear1 != nYear2)                                 //如果年不相等，直接得出了日期的大小
    {
        return nYear1 - nYear2;
    }
    //比较月
    int nMonth1 = atoi(strDate1.Mid(5, 2));
    int nMonth2 = atoi(strDate2.Mid(5, 2));
    if(nMonth1 != nMonth2)                                //如果年相等，月不相等，得出日期的大小
    {
        return nMonth1 = nMonth2;
    }
    //比较日
    int nDay1 = atoi(strDate1.Mid(8, 2));
    int nDay2 = atoi(strDate2.Mid(8, 2));
    if(nDay1 != nDay2)                                    //年和月相等，进行日的比较
    {
        return nDay1 - nDay2;
    }
    return 0;                                             //日期相等，返回 0
}
```

（16）向 CSortListCtrl 类中添加静态成员函数 SortFunction，用于进行列排序。这里需要注意的是函数的前两个参数，这两个参数将作为两个视图项关联数据，如果视图项没有数据关联，则参数为 0，之前在加载视图项时，会为视图项关联一个 CItemData 结构数据，其作用就在于此。代码如下：

```
int CALLBACK CSortListCtrl::SortFunction(LPARAM lParam1, LPARAM lParam2, LPARAM lParamData)
{
    CSortListCtrl* pListCtrl = (CSortListCtrl*)(lParamData);
    CItemData* pParam1 = (CItemData*)(lParam1);
    CItemData* pParam2 = (CItemData*)(lParam2);
    //获取排序列的文本
    LPCTSTR pszText1 = pParam1->m_ColumnTexts[pListCtrl->m_nSortColumn];
    LPCTSTR pszText2 = pParam2->m_ColumnTexts[pListCtrl->m_nSortColumn];
    if( IsNumber( pszText1 ) )                            //按数值比较
        return pListCtrl->m_bAscend ? CompareDataAsNumber(pszText1, pszText2) :
            CompareDataAsNumber(pszText2, pszText1);
    else if(IsDate(pszText1))                             //按日期比较
        return pListCtrl->m_bAscend ? CompareDataAsDate(pszText1, pszText2) :
            CompareDataAsDate(pszText2, pszText1);
    else                                                  //按文本比较
        return pListCtrl->m_bAscend ? lstrcmp(pszText1, pszText2) : lstrcmp(pszText2, pszText1);
}
```

（17）向 CSortListCtrl 类中添加 Sort 方法，调用列表视图的 SortItems 方法对某一列进行升序或降序排序。代码如下：



```
void CSortListCtrl::Sort(int iColumn, BOOL bAscending)
{
	m_nSortColumn = iColumn;                              //设置排序列
	m_bAscend = bAscending;                               //设置升序后降序排列
	m_ctlHeader.SetSortColomn(m_nSortColumn, m_bAscend);
	SortItems(SortFunction, (DWORD)this);                 //进行列排序
}
```

（18）处理列表视图控件列的单击事件，当用户单击某一列时，对列进行排序。代码如下：

```
void CSortListCtrl::OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
	int nColumn = pNMListView->iSubItem;                  //获取当前列
	//对列进行排序
	Sort(nColumn, nColumn == m_nSortColumn ? !m_bAscend : TRUE);
	*pResult = 0;
}
```

（19）改写列表视图控件的 PreSubclassWindow 方法，将自定义的列表控件关联到列表视图控件中。代码如下：

```
void CSortListCtrl::PreSubclassWindow()
{
	ASSERT(GetStyle() & LVS_REPORT);
	CListCtrl::PreSubclassWindow();
	VERIFY(m_ctlHeader.SubclassWindow(GetHeaderCtrl()->GetSafeHwnd()));
}
```

至此，完成了列表视图控件主要功能的设计（在列表视图控件销毁时需要释放项目关联的数据结构，由于篇幅限制，这里没有给出相关代码，详细代码请参考资源包中的程序）。

7.4.2　登录 FTP 服务器

在进行与 FTP 有关的操作前，首先需要登录 FTP 服务器才能进行其他相关操作。在 MFC 中提供了 CInternetSession 类用于连接网络服务器。通过调用 CInternetSession 类的 GetFtpConnection 方法可以连接 FTP 服务器，获取一个与 FTP 服务器关联的 CFtpConnection 对象指针。主要代码如下：

```
CString csServer,csPassword,csUser,csPort;
m_ConnectBar.GetDlgItemText(IDC_FTPSERVER,csServer);       //获取 FTP 服务器 IP
m_ConnectBar.GetDlgItemText(IDC_FTPPORT,csPort);           //获取 FTP 服务器端口
m_ConnectBar.GetDlgItemText(IDC_PASSWORD,csPassword);      //获取登录密码
m_ConnectBar.GetDlgItemText(IDC_USER,csUser);              //获取用户名
if (!csServer.IsEmpty() && !csPassword.IsEmpty() &&
		!csUser.IsEmpty() && !csPort.IsEmpty())            //判断登录信息是否为空
{
```

```
    try
    {
        //开始登录服务器
        m_pFtp =  Session.GetFtpConnection(csServer,csUser,csPassword,atoi(csPort));
        m_bLoginSucc = TRUE;
    }
    catch(CInternetException &e)                            //进行异常捕捉
    {
        m_bLoginSucc = FALSE;                               //登录失败
        delete m_pFtp;
    }
    delete m_pFtp;                                          //释放 FTP 连接对象
}
```

7.4.3 实现 FTP 目录浏览

在登录服务器之后，需要将 FTP 服务器上的目录和文件信息显示在本地的窗口中，这就需要实现浏览 FTP 目录。

为了查找 FTP 服务器上的目录和文件，MFC 提供了 CFtpFileFind 类，该类与 CFileFind 类的作用、用法几乎相同，使用 FindFile 方法开始查找一个文件，使用 FindNextFile 方法查找下一个文件。只是 CFtpFileFind 类用于搜索 FTP 服务器。此外，在定义一个 CFtpFileFind 类对象时，需要关联一个 FTP 服务器的连接。下面给出实现 FTP 目录浏览的关键代码。

```
CFtpConnection* pFTP = NULL;                                //定义 FTP 连接对象指针
//登录 FTP 服务器
pFTP = m_Session.GetFtpConnection(m_FtpServer,m_User,m_Password,atoi(m_Port));
pFTP->SetCurrentDirectory("");                              //设置当前目录
CFtpFileFind Find(pFTP);                                    //定义 FTP 文件查找对象
BOOL bFind;                                                 //记录查找结果
m_CurDir =lpPath;                                           //设置当前目录
if (strlen(lpPath)==0)
    bFind = Find.FindFile(NULL,INTERNET_FLAG_EXISTING_CONNECT|
        INTERNET_FLAG_RELOAD);                              //从 FTP 根目录开始查找
else
    bFind = Find.FindFile(lpPath,INTERNET_FLAG_EXISTING_CONNECT|
        INTERNET_FLAG_RELOAD);                              //查找指定目录
if (bFind)                                                  //查找是否成功
{
    CString csFileName,csDataTime,csFileSize;
    while (bFind)                                           //遍历当前目录
    {
        bFind = Find.FindNextFile();                        //查找下一个文件
        csFileName = Find.GetFileName();                    //获取文件名称
        CTime fileTime;
        Find.GetLastWriteTime(fileTime);                    //获取文件修改时间
```



```
//格式化文件修改时间
csDataTime = fileTime.Format("%Y-%m-%d %H:%M");
if (!Find.IsDots() && !Find.IsHidden())
{
	__int64 lFileLen = Find.GetLength64();			//获取文件长度
	if (Find.IsDirectory())					//判断是否为目录
	{
		csFileSize = "文件夹";
	}
	else							//如果是文件
	{
		double fGB = lFileLen /(double)(1024*1024*1024);
		if (fGB < 1)					//是否小于 1GB
		{
			double fMB = lFileLen / (double)(1024*1024);
			if (fMB < 1)				//是否小于 1MB
			{
				double fBK = lFileLen / (double)(1024);
				if (fBK >1)			//是否小于 1KB
				{
					csFileSize.Format("%2.2f KB",fBK);
				}
				else				//以位为单位显示
				{
					csFileSize.Format("%i B",lFileLen);
				}
			}
			else
			{
				csFileSize.Format("%2.2f MB",fMB);
			}
		}
		else
		{
			csFileSize.Format("%2.2f GB",fGB);
		}
	}
}
int nItem = AddItem(csFileName,csFileSize,csDataTime); //添加视图项
//设置文件显示的图标
SHFILEINFO shInfo;						//定义文件外壳信息
int nIcon = 0;
CString csName = Find.GetFileName();				//获取文件名
int nPos = csName.ReverseFind('.');				//查找扩展名
if (nPos >0)							//如果是文件
{
	//映射为本地系统的文件图标
	SHGetFileInfo(csName,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
```

```
                    SHGFI_ICON | SHGFI_SMALLICON|SHGFI_USEFILEATTRIBUTES);
        }
        else                                                    //如果是目录
        {
            char chPath[MAX_PATH] = {0};
            GetCurrentDirectory(MAX_PATH,chPath);               //获取当前目录
            SHGetFileInfo(chPath,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
                SHGFI_ICON | SHGFI_SMALLICON);                  //获取目录图标
        }
        DestroyIcon(shInfo.hIcon);
        nIcon = shInfo.iIcon;                                   //获取图标索引
        SetItem(nItem,0,LVIF_IMAGE,"",nIcon,0,0,0);             //设置视图项显示的图标
         //设置项目标记，0 表示文件，1 表示目录
        if (Find.IsDirectory())
        {
            SetItemData(nItem,1);                               //设置视图项标记
        }
        else
        {
            SetItemData(nItem,0);                               //设置视图项标记
        }
    }
}
Find.Close();                                                   //关闭文件查找对象
pFTP->Close();                                                  //关闭 FTP 文件连接
delete pFTP;                                                    //释放 FTP 连接对象
```

7.4.4 多任务下载 FTP 文件

在设计 FTP 文件传输管理软件时，为了实现多任务下载，并能够对每个下载任务进行控制，采用了单独的线程来下载文件，每一个线程维护一个下载任务。在任务执行过程中，用户可以通过挂起线程、唤醒线程来执行暂停、继续任务，通过结束线程来取消任务。

为了能够在任务列表中执行挂起、唤醒和终止线程操作，需要为每一个任务关联一个线程句柄，笔者采用的方式是为视图项关联一个额外的整数值（线程句柄）。利用列表视图控件的 SetItemData 方法将线程句柄设置为视图项的额外数据。

```
m_pTastView->m_TastList.SetItemData(nCurItem,(DWORD)hHandle);
```

为了进行多线程下载，需要单独定义一个文件下载的函数。当用户下载一个任务时，可以创建一个线程，在线程函数中调用该函数来执行下载任务。该函数需要实现下载指定的文件，或者下载指定目录下的所有文件，从功能描述可知，下载函数将被设计为递归函数。下面给出下载函数的代码。

```
/**************************************参数说明*******************************************
pDlg:           表示主窗口，在这里没有实际意义
lpDir:          表示当前下载的文件或目录
```



```
csRelativePath:表示文件存储到本地的相对路径，与 lpSaveDir 参数构成完整的路径
nFlag:          表示结束标记，如果用户取消了下载任务，立即退出 DownLoadFile 函数
bIsFile:        表示当前的文件名是否是一个完整文件名，在首次调用 DownLoadFile 函数时，
                bIsFile 为 TRUE。例如，当下载的 FTP 目录为 Data，如果 Data 目录下的 zone.txt 文件是最后一
                个查找的文件，则查找该文件时获得的文件名是 Data/zone.txt，该文件名是一个完整的文
                件名，不需要再添加父目录
lpServer:       表示 FTP 服务器地址
lpUser:         表示登录用户名
lpPassword:     表示登录密码
nPort:          表示端口号
lpSaveDir:      表示下载文件在本地的根存储路径
*****************************************************************************************/
void CMainFrame::DownLoadFile(CMainFrame * pDlg,LPCTSTR lpDir,LPCTSTR csRelativePath,
DWORD nFlag, BOOL bIsFile,LPCTSTR lpServer,LPCTSTR lpUser,LPCTSTR lpPassword,
    int nPort,LPCTSTR lpSaveDir)
{
    CFtpConnection* pTemp = NULL;                           //定义一个临时的 FTP 连接对象
    //连接 FTP 服务器
    pTemp =   Session.GetFtpConnection(lpServer,lpUser,lpPassword,nPort);
    CFtpFileFind Find(pTemp);                               //定义文件查找对象
    CString   filename;
    if (m_dwStop == nFlag)                                  //表示当前任务被挂起
    {
        pTemp->Close();                                     //关闭连接
        delete pTemp;                                       //释放连接对象
        Find.Close();                                       //关闭文件查找
        return;
    }
    BOOL ret;
    if (strlen(lpDir)==0)                                   //从 FTP 根目录开始查找
        ret = Find.FindFile(NULL,INTERNET_FLAG_EXISTING_CONNECT);
    else                                                    //查找指定目录
        ret = Find.FindFile(lpDir,INTERNET_FLAG_EXISTING_CONNECT);
    if (ret)
    {
        while (Find.FindNextFile())                         //查找下一个文件
        {
            filename = Find.GetFileName();                  //获取文件名称
            if (Find.IsDirectory())                         //如果文件是目录，递归调用 DownLoadFile
            {
                char csdir[MAX_PATH] = {0};
                char csRetPath[MAX_PATH] = {0};
                strcpy(csdir,lpDir);
                strcat(csdir,"/");
                strcat(csdir,filename);                     //设置查找目录
                strcpy(csRetPath,csRelativePath);           //设置本地存储路径
                strcat(csRetPath,"/");
                strcat(csRetPath,filename);
```

```
        DownLoadFile(pDlg,csdir,csRetPath,nFlag,false,lpServer,lpUser,
            lpPassword,nPort,lpSaveDir);                //先遍历子目录
    }
    else                                                //如果是文件，则下载到本地
    {
        char csUrl[MAX_PATH] = {0};
        strcpy(csUrl,lpDir);
        strcat(csUrl,"/");
        strcat(csUrl,Find.GetFileName());               //获取文件的 FTP 完整路径
        filename = Find.GetFileName();                  //获取文件名
        //打开 FTP 服务器上的文件
        CInternetFile* pFile = pTemp->OpenFile(csUrl,GENERIC_READ,INTERNET
                _FLAG_TRANSFER_BINARY|INTERNET_FLAG_RELOAD);
        if (pFile!=NULL)
        {
            DWORD dwLen = Find.GetLength();             //获取文件长度
            DWORD dwReadNum = 0;
            CFile file;                                 //定义一个文件对象
            CString csDir = csRelativePath;             //设置存储路径
            csDir.Replace('/','\\');
            csDir = lpSaveDir + csDir +"\\";
            pDlg->MakeSureDirectoryPathExists(csDir);   //先创建目录
            csDir += filename;                          //设置下载到本地的文件名
            //创建文件
            file.Open(csDir,CFile::modeCreate|CFile::modeWrite);
            void *pBuffer =   NULL;
            //在堆中分配空间
            HGLOBAL hHeap = GlobalAlloc(GMEM_FIXED|GMEM_ZEROINIT,8192);
            pBuffer = GlobalLock(hHeap);                //锁定堆空间
            int nNum = 1;
            while (dwReadNum < dwLen)                   //循环读取文件数据
            {
                if (m_dwStop == nFlag)                  //表示当前任务被挂起
                {
                    pTemp->Close();                     //关闭 FTP 连接
                    delete pTemp;                       //释放连接对象
                    pFile->Close();                     //关闭 FTP 文件
                    file.Close();                       //关闭本地文件
                    Find.Close();                       //关闭文件查找对象
                    delete pFile;                       //释放 FTP 文件
                    GlobalUnlock(hHeap);                //解锁堆空间
                    GlobalFree(hHeap);                  //释放堆空间
                    return;                             //退出函数
                }
                nNum ++;
                UINT nFact = pFile->Read(pBuffer,8192); //读取文件数据到堆中
                if (nFact > 0)
                {
```

```
                    file.Write(pBuffer,nFact);          //向文件中写入数据
                }
                dwReadNum += nFact;                     //记录接收的文件数量
            }
            GlobalUnlock(hHeap);                        //解锁堆空间
            GlobalFree(hHeap);                          //释放堆空间
            pFile->Close();                             //关闭 FTP 文件
            delete pFile;                               //释放 FTP 文件
            file.Close();                               //关闭打开的文件
        }
    }
}
//在文件查找后，下载最后一个文件或目录
if (Find.IsDirectory())                                 //判断是否为目录
{
    filename = Find.GetFileName();                      //获取文件名
    char csdir[MAX_PATH] = {0};
    char csRetPath[MAX_PATH] = {0};
    strcpy(csdir,lpDir);
    strcat(csdir,"/");
    strcat(csdir,filename);                             //设置查找子目录
    strcpy(csRetPath,csRelativePath);
    strcat(csRetPath,"/");
    strcat(csRetPath,filename);                         //设置本地存储路径
        DownLoadFile(pDlg,csdir,csRetPath,nFlag,false,lpServer,lpUser,
            lpPassword,nPort,lpSaveDir);                //递归调用 DownLoadFile 函数
}
else                                                    //如果是文件
{
    char csUrl[MAX_PATH] = {0};
    //当前的文件名是否是一个完整文件名，在首次调用 DownLoadFile 函数时 bIsFile 为 TRUE
    //例如，当下载的 FTP 目录为 Data，如果 Data 目录下的 zone.txt 文件是最后一个查找的文件
    //则查找该文件时获得的文件名是 Data/zone.txt，该文件名是一个完整的文件名
    //不需要再添加父目录
    if(bIsFile==FALSE)                                  //文件名包含一个相对路径
    {
        strcpy(csUrl,lpDir);                            //设置完整的文件名
        strcat(csUrl,"/");
        strcat(csUrl,Find.GetFileName());
    }
    else                                                //文件名是一个完整文件名
    {
        strcpy(csUrl,lpDir);                            //直接复制文件名
    }
    filename = Find.GetFileName();                      //获取文件名
    //打开 FTP 文件
    CInternetFile* pFile = pTemp->OpenFile(csUrl ,GENERIC_READ,
            INTERNET_FLAG_TRANSFER_BINARY|INTERNET_FLAG_RELOAD);
```

```
        if (pFile!=NULL)
        {
            DWORD dwLen = Find.GetLength();                 //获取文件长度
            DWORD dwReadNum = 0;
            CFile file;                                     //定义文件对象
            CString csDir;
            if (bIsFile==FALSE)                             //文件名中包含相对路径
            {
                csDir = csRelativePath;
                csDir += "/";
                csDir += Find.GetFileName();
                csDir.Replace('/','\\');
                csDir = lpSaveDir + csDir;                  //设置下载到本地的路径
                CString csPath;
                int nPos = csDir.ReverseFind('\\');
                csPath = csDir.Left(nPos+1);
                pDlg->MakeSureDirectoryPathExists(csPath);  //先创建目录
            }
            else                                            //如果文件名中包含完整路径
            {
                csDir = lpSaveDir;
                csDir += "\\";
                csDir += csRelativePath;                    //设置下载到本地的路径
            }
            //创建文件
            file.Open(csDir,CFile::modeCreate|CFile::modeWrite);
            void *pBuffer = NULL;
            //在堆中分配空间
            HGLOBAL hHeap = GlobalAlloc(GMEM_FIXED|GMEM_ZEROINIT,8192);
            pBuffer = GlobalLock(hHeap);                    //锁定堆空间
            int nNum = 1;
            while (dwReadNum < dwLen)                       //利用循环写入文件数据
            {
                if (m_dwStop == nFlag)                      //表示当前任务被挂起
                {
                    pTemp->Close();                         //关闭 FTP 连接
                    delete pTemp;                           //释放 FTP 连接
                    pFile->Close();                         //关闭 FTP 文件
                    file.Close();                           //关闭本地文件
                    Find.Close();                           //关闭文件查找对象
                    delete pFile;                           //释放 FTP 文件对象
                    GlobalUnlock(hHeap);                    //解锁堆空间
                    GlobalFree(hHeap);                      //释放堆空间
                    return;
                }
                UINT nFact = pFile->Read(pBuffer,8192);     //读取文件数据
                if (nFact >0)
                {
```

```
                        file.Write(pBuffer,nFact);          //向文件中写入数据
                    }
                    dwReadNum += nFact;                     //累加写入的数量
                }
                GlobalUnlock(hHeap);                        //解锁堆空间
                GlobalFree(hHeap);                          //释放堆空间
                pFile->Close();                             //关闭 FTP 文件
                delete pFile;                               //释放 FTP 文件对象
                file.Close();                               //关闭本地文件
            }
        }
    }
    Find.Close();                                           //关闭文件查找对象
    delete pTemp;                                           //释放 FTP 连接
}
```

有了下载函数，就可以在线程函数中调用该函数实现文件下载。但是，在线程函数中需要提供有关下载的信息，例如下载的文件名、下载文件在本地系统存储的路径、线程对应任务列表中的视图项、线程句柄等。为此，需要单独定义一个线程参数结构，代码如下：

```
struct ThreadParam
{
    CMainFrame* pDlg;                                       //当前对话框
    int nItem;                                              //线程对应列表中的任务
    char m_DownFile[MAX_PATH];                              //下载文件
    char m_RelativeFile[MAX_PATH];                          //下载到本地的路径
    int nDownFlag;                                          //下载标记
    HANDLE m_hThread;                                       //线程句柄
};
```

下面给出下载文件使用的线程函数，在线程函数中主要是调用 DownLoadFile 方法实现文件下载。代码如下：

```
//下载文件的线程函数
DWORD __stdcall DownloadThreadProc(LPVOID lpParameter)
{
    ThreadParam *Param = (ThreadParam *)lpParameter;        //获取线程参数
    CMainFrame * pDlg = Param->pDlg;                        //获取主对话框指针
    int nItem = Param->nItem;                               //获取当前下载任务对应的视图项索引
    char downfile[MAX_PATH] = {0};
    strcpy(downfile,Param->m_DownFile);                     //复制下载文件名
    char relfile[MAX_PATH] = {0};
    strcpy(relfile,Param->m_RelativeFile);                  //复制相对路径
    pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"正在下载");
    if (Param->nDownFlag ==0)                               //当前选择的是文件
    {
        //调用 DownLoadFile 方法下载文件
        pDlg->DownLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,true,pDlg->m_csServer,
```

```
                pDlg->m_csUser,pDlg->m_csPassword,pDlg->m_nPort,pDlg->m_csDownDir);
    }
    else if(Param->nDownFlag ==1)
    {
        //调用 DownLoadFile 方法下载文件

    pDlg->DownLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,false,pDlg->m_csServepDlg->
                m_csUser,pDlg->m_csPassword,pDlg->m_nPort,pDlg->m_csDownDir);
    }
    pDlg->m_pTastView->m_TastList.SetItemText(nItem,3,"完成");
    pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"下载完成");
    if (pDlg->m_dwStop == (DWORD)Param->m_hThread)          //终止线程后设置初始标记
    {
        pDlg->m_dwStop = 0;                                 //恢复任务终止标记
    }
    //任务完成后删除任务列表中对应的视图项
    pDlg->DeleteItemFormData(&pDlg->m_pTastView->m_TastList,(DWORD)Param->m_hThread);
    int nCount = pDlg->m_pTastView->m_TastList.GetItemCount();
    delete Param;                                           //释放线程参数
    if (pDlg->m_bTurnOff && nCount == 0)                    //下载后是否关机
    {
        pDlg->TurnOff();                                    //关机操作
    }
    return 0;
}
```

有了下载函数和下载使用的线程函数，用户在下载文件时只需要填充线程参数，然后创建一个线程即可。例如，使用下面的代码开始一个下载任务。代码如下：

```
ThreadParam * Param = new ThreadParam();                    //创建线程参数
Param->pDlg = this;                                         //填充线程参数
Param->nItem = nCurItem;
Param->nDownFlag =m_pFtpView->m_RemoteFiles.GetItemData(nItem);
memset(Param->m_DownFile,0,MAX_PATH);
memset(Param->m_RelativeFile,0,MAX_PATH);
strcpy(Param->m_DownFile, m_pFtpView->m_RemoteFiles.m_CurDir);
strcat(Param->m_DownFile,text);
strcpy(Param->m_RelativeFile,text);
//创建新的线程，执行线程函数
HANDLE hHandle = CreateThread(NULL,0,DownloadThreadProc,(void*)Param,0,NULL);
m_pTastView->m_TastList.SetItemData(nCurItem,(DWORD)hHandle);
```

7.4.5 在任务列表中暂停、取消某一任务

在 FTP 文件传输管理软件中，当任务列表中同时存在多个任务时，用户可以暂停或取消某一个任务。对于任务的暂停，实现起来比较简单，只需要挂起线程即可，因为任务列表中的每一个视图项（一行数据）关联一个整数值，即线程句柄，有了线程句柄，就可以挂起线程。代码如下：



```
void CTastListView::OnBtStop()
{
	int nSel = m_TastList.GetSelectionMark();				//获取当前任务的视图项
	if (nSel != -1)
	{
		DWORD nItemData = m_TastList.GetItemData(nSel);		//读取任务对应的线程句柄
		CString csState = m_TastList.GetItemText(nSel,3);
		if (csState != "暂停")							//任务是否已挂起
		{
			SuspendThread((HANDLE)nItemData);				//挂起线程
			m_TastList.SetItemText(nSel,3,"暂停");
		}
	}
}
```

这里有一点需要注意的是，在挂起线程时需要判断线程是否已挂起，如果没有则挂起，否则不执行操作。使用判断的目的是如果一个线程已挂起，再进行挂起操作，则在唤醒线程时需要执行相同次数的唤醒操作。

而对于取消任务的实现就没有那么简单了。读者也许会认为，只要结束了线程，下载任务也就取消了，方法应该与线程的暂停没有什么差别。大家不要忘记，终止一个线程的最好方式是线程函数的结束，如果使用 ExitThread、TerminateThread 之类的函数来“意外”终止一个线程，会导致线程函数的堆栈不能释放，很容易出现内存泄露。例如，在线程函数中分配一个堆空间，此时线程函数结束，那么在线程函数中其后释放堆空间的代码就不会被执行，导致内存泄露。

笔者采用的方式是在主对话框中定义一个变量 m_dwStop，在线程函数中判断如果 m_dwStop 与当前的线程句柄相同，表示用户取消的任务，则恢复 m_dwStop 的初始值为零。正在进行取消任务操作的代码位于下载文件的函数中，在下载函数 DownLoadFile 中，多处出现类似下面的代码。

```
if (m_dwStop == nFlag)								//表示当前任务被挂起
{
	pTemp->Close();								//关闭 FTP 连接
	delete pTemp;									//释放 FTP 连接对象
	Find.Close();									//关闭文件查找对象
	return;
}
```

上述代码用于判断是否结束任务，如果是，则立即退出下载函数 DownLoadFile，线程函数立即结束执行，这样通过提前退出线程函数实现了线程的终止，也就实现了任务的取消。

7.4.6　利用鼠标拖曳实现文件的上传/下载

在 FTP 文件传输管理软件中，为了方便用户操作，提供了对鼠标拖曳操作的支持。用户可以利用鼠标直接将某一个本地文件拖放到 FTP 服务器文件中，实现文件的上传，也可以利用鼠标将 FTP 服务器上的文件拖放到本地信息的列表视图控件中，实现文件的下载。效果如图 7.9 所示。

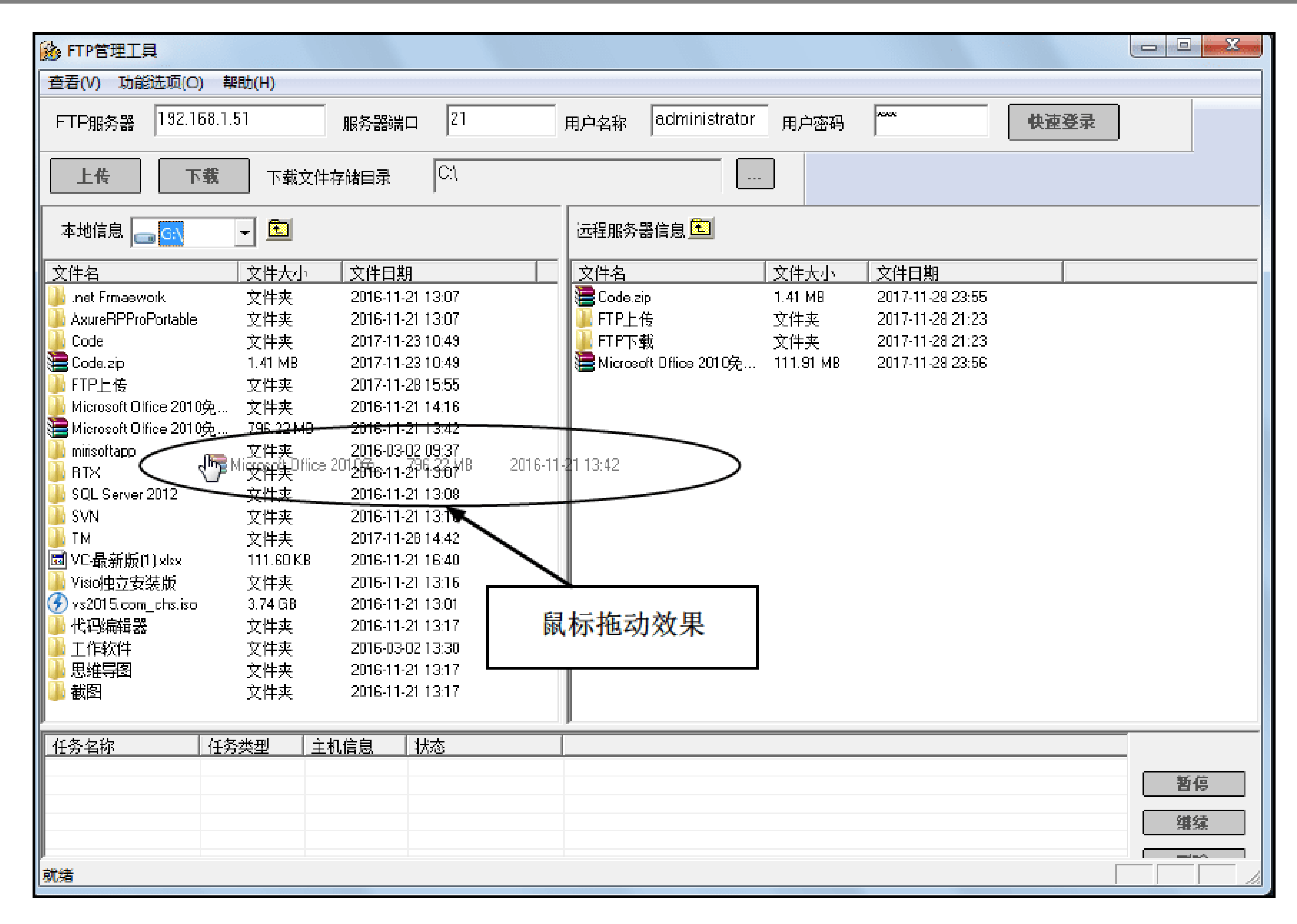


图 7.9　鼠标拖曳效果

对于列表视图控件来说，实现鼠标拖曳的效果比较简单，主要分为 3 个步骤，第一步，处理列表视图控件的 LVN_BEGINDRAG 消息，开始鼠标拖曳。代码如下：

```
void CFTPView::OnBegindragRemotefiles(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
	*pResult = 0;
	int nItem=pNMListView->iItem;                              //获取拖动的视图项
	POINT pt=pNMListView->ptAction;                            //获取当前坐标
	m_pDragList = m_RemoteFiles.CreateDragImage(nItem,&pt);    //创建一个拖动图像列表
	m_pDragList->BeginDrag(0,CPoint(0,0));                     //开始拖动
	m_pDragList->DragEnter(NULL,pt);
	SetCapture();                                              //捕捉鼠标
	m_bDrag = TRUE;                                            //设置拖动标记
	*pResult = 0;
}
```

第二步，在鼠标移动过程中设置图像拖动的位置。代码如下：

```
void CFTPView::OnMouseMove(UINT nFlags, CPoint point)
{
	if (m_bDrag)                                               //拖动过程中
	{
		CRect rect;
		GetWindowRect(&rect);                                  //获取窗口区域
		ClientToScreen(&point);                                //转换坐标
```

```
        m_pDragList->DragMove(point);                //沿着鼠标拖动图像
    }
    CFormView::OnMouseMove(nFlags, point);
}
```

最后一步，在释放鼠标按钮时结束拖曳。如果拖放到目标窗口，则执行相关操作，本例为执行文件下载操作。代码如下：

```
void CFTPView::OnLButtonUp(UINT nFlags, CPoint point)
{
    if(m_bDrag)
    {
        m_bDrag = FALSE;                             //设置拖动结束标记
        CImageList::DragLeave(NULL);                 //释放鼠标拖动
        CImageList::EndDrag();
        ReleaseCapture();                            //释放鼠标捕捉
        delete m_pDragList;                          //释放拖动的图像列表
        m_pDragList=NULL;
        CRect rect;
        CMainFrame * pDlg = (CMainFrame*)AfxGetMainWnd();
        pDlg->m_pLocalView->m_LocalFiles.GetWindowRect(&rect);
        ClientToScreen(&point);                      //转换客户坐标
        if(rect.PtInRect(point))                     //判断是否拖放到本地列表视图中
        {
            pDlg->OnDownload();                      //执行下载操作
        }
    }
    CFormView::OnLButtonUp(nFlags, point);
}
```

7.4.7　直接创建多级目录

在下载 FTP 文件时，需要将 FTP 文件保存到本地系统指定的目录下，并且需要保持文件在 FTP 服务器上的目录结构。例如，下载 FTP 服务器上的 Files 目录下的所有文件，需要将 Files 目录下的所有文件和目录保持原来的目录结构下载到本地的某一目录下。这经常涉及到在多级目录下创建文件，如果目录不存在，将导致创建文件失败。为了防止文件创建失败，在创建文件前需要先创建文件目录，该目录可能是一个多级目录。使用 CreateDirectory 函数只能创建一级目录，如果创建多级目录，需要循环调用 CreateDirectory 函数。为了能够创建多级目录，可以使用 MakeSureDirectoryPathExist 函数，该函数位于 dbghelp.dll 动态链接库中，在开发环境下没有进行封装，在使用前需要导入该函数。首先定义一个函数指针类型。代码如下：

```
typedef BOOL(__stdcall funMakeSure)(PCSTR DirPath);
```

然后定义一个函数指针。代码如下：

```
funMakeSure * MakeSureDirectoryPathExists;                   //定义函数指针
```



最后加载 dbghelp.dll 动态链接库，获取 MakeSureDirectoryPathExist 函数地址。代码如下：

```
HMODULE hMod = LoadLibrary("dbghelp.dll");
MakeSureDirectoryPathExists = (funMakeSure *)GetProcAddress(hMod,"MakeSureDirectoryPathExists");
```

在获取 MakeSureDirectoryPathExist 函数地址后就可以使用 MakeSureDirectoryPathExists 函数创建多级目录。代码如下：

```
MakeSureDirectoryPathExists(“c:\\database\\sqlserver\\data”);
```

7.4.8　根据文件扩展名获取文件的系统图标

在设计 FTP 文件传输管理软件时，当浏览 FTP 服务器目录时，需要将 FTP 服务器上的目录和文件以本地的系统图标表示。为此，需要实现根据文件的类型，也就是文件扩展名获取文件的系统图标。

根据文件类型获取文件图标，可以通过搜索注册表来实现，文件类型的图标在注册表中可以找到，但是实现起来相对比较麻烦。笔者经过测试，可以通过使用 SHGetFileInfo 函数实现该功能。代码如下：

```
//根据文件类型获取图标
HICON CSortListCtrl::GetIconFromExtendedName(LPCTSTR lpName)
{
	SHFILEINFO shInfo;                                    //定义外壳文件信息
	int nIcon = 0;
	CString extension = lpName;
	CString csName = "text"+extension;                    //设置一个临时的文件名
	int nPos = csName.ReverseFind('.');                   //判断是否为文件
	if (nPos >0)
	{
		//获取文件图标
		SHGetFileInfo(csName,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
			SHGFI_ICON | SHGFI_SMALLICON|SHGFI_USEFILEATTRIBUTES);
	}
	else                                                  //参数表示一个目录
	{
		char chPath[MAX_PATH] = {0};
		GetCurrentDirectory(MAX_PATH,chPath);             //获取当前目录
		SHGetFileInfo(chPath,FILE_ATTRIBUTE_NORMAL,&shInfo,sizeof(shInfo),
			SHGFI_ICON | SHGFI_SMALLICON);                //获取目录图标
	}
	nPos = shInfo.iIcon;
	return shInfo.hIcon;                                  //返回获取的图标
}
```

上述代码中应注意 SHGFI_USEFILEATTRIBUTES 标记，该标记不通过路径访问文件，而假设文件已存在，这样，返回的就是该文件类型的图标。



7.4.9　关闭工具栏时取消菜单项的复选标记

在创建一个单文档/视图结构的应用程序时，如果用户将工具栏拖出框架窗口，然后将其关闭，则“查看”菜单下的“工具栏”菜单项的复选标记自动取消，效果如图 7.10 所示。

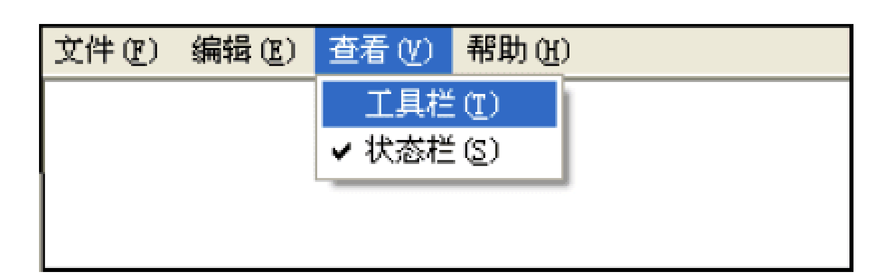


图 7.10　取消菜单项的复选标记

在设计 FTP 文件传输管理软件时，也需要实现类似的功能。当用户将登录信息栏拖出框架并关闭时，“查看”菜单下的“登录信息栏”按钮复选标记自动取消。为了实现该功能，需要在创建登录信息栏时将信息栏的 ID 设置为对应的菜单项 ID。例如：

```
if (!m_ConnectBar.Create(this, IDD_CONNECTFTPDLG, CBRS_TOP |
    CBRS_TOOLTIPS | CBRS_FLYBY|CBRS_SIZE_DYNAMIC, ID_CONNECTBAR))
{
    TRACE0("Failed to create Dialog bar\n");
    return -1;
}
```

然后处理菜单项的 UPDATE-COMMAND_UI 消息，根据信息栏是否可见设置或取消菜单项的复选标记。代码如下：

```
void CMainFrame::OnUpdateLoginbar(CCmdUI* pCmdUI)
{
    CControlBar* pBar = GetControlBar(pCmdUI->m_nID);              //获取控制栏
    if (pBar != NULL)
    {
        pCmdUI->SetCheck((pBar->GetStyle() & WS_VISIBLE) != 0);   //设置控制栏是否可见
        return;
    }
    pCmdUI->ContinueRouting();                                      //继续传递消息
}
```

7.5　主窗口设计



7.5.1　主窗口概述

FTP 文件传输管理软件主窗口由菜单、登录信息栏、工具栏、本地信息窗口、远程 FTP 服务器信息窗口和任务列表窗口 6 个部分构成，效果如图 7.11 所示。



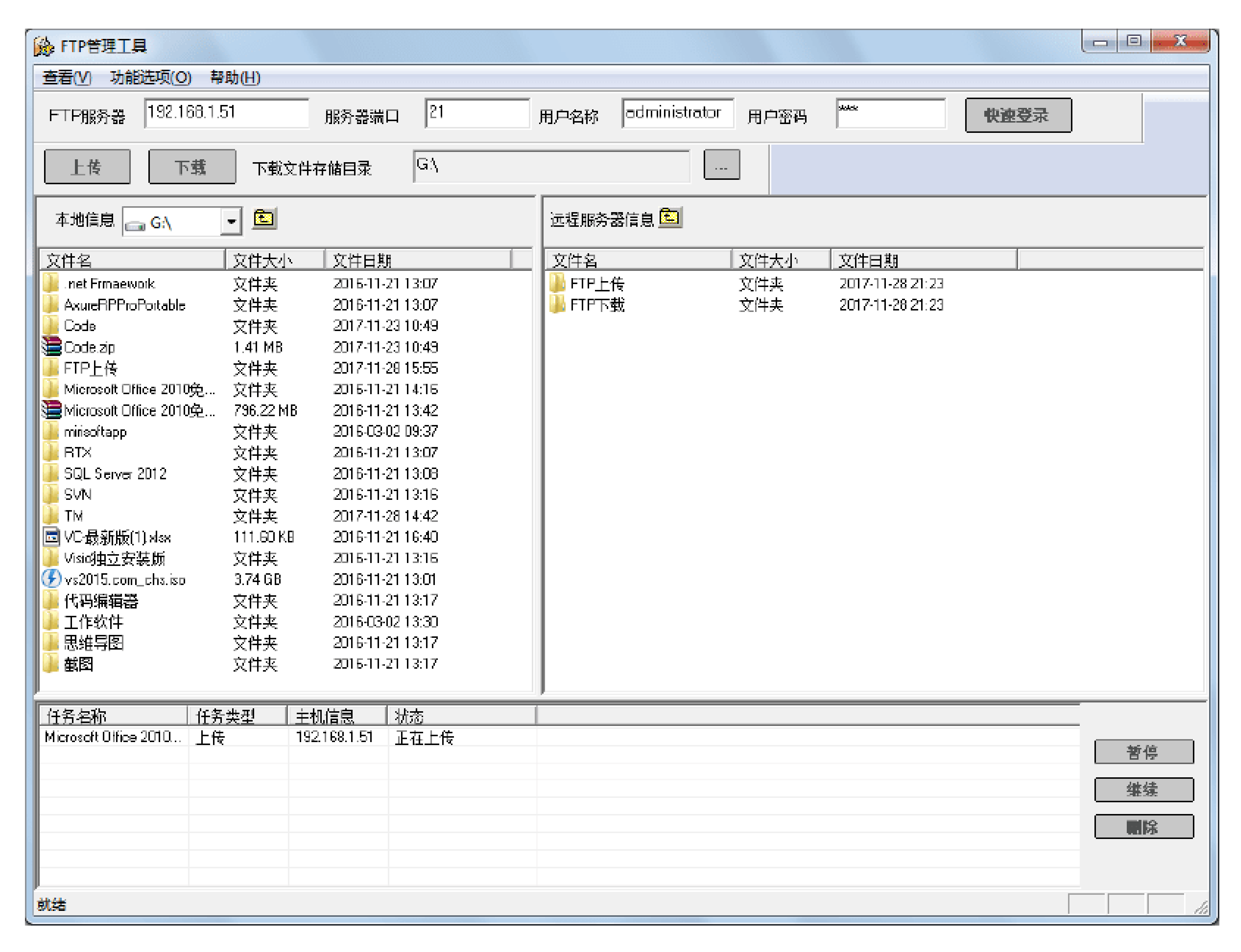


图 7.11　FTP 文件传输管理软件主窗口

7.5.2　主窗口界面布局

FTP 文件传输管理软件主窗口主要由控制条和视图窗口构成。窗口的布局都是通过代码控制的，没有涉及设计期的控件放置和属性设置操作。这里笔者主要介绍一下视图窗口的布局，视图窗口主要分为 3 个部分，分别为本地信息窗口、远程 FTP 服务器视图窗口和任务列表窗口。这 3 个窗口是通过视图分割将其嵌入到框架中的。下面给出视图分割的主要代码。

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    if (!m_wndSplitter.CreateStatic(this, 2, 1))                //将主窗口分割为 2 行 1 列
    {
        TRACE0("Failed to create splitter bar ");
        return FALSE;
    }
    if (!m_ChildSplitter.CreateStatic(&m_wndSplitter, 1, 2,WS_CHILD|WS_VISIBLE,
        m_wndSplitter.IdFromRowCol(0,0)))                        //将主窗口第 1 行分割为 2 列
    {
        TRACE0("Failed to create splitter bar ");
        return FALSE;
    }
    //在第 1 行第 1 列填充视图窗口
    m_ChildSplitter.CreateView(0,0,RUNTIME_CLASS(CLocalView),CSize(200,180),pContext);
    //在第 1 行第 2 列填充视图窗口
    m_ChildSplitter.CreateView(0,1,RUNTIME_CLASS(CFTPView),CSize(180,180),pContext);
```



```
    //在第 2 行填充视图窗口
    m_wndSplitter.CreateView(1,0,RUNTIME_CLASS(CTastListView),CSize(0,10),pContext);
    //设置第 1 行的高度
    m_wndSplitter.SetRowInfo(0,400,50);
    //设置第 1 行各列的宽度
    m_ChildSplitter.SetColumnInfo(0,400,50);
    m_ChildSplitter.SetColumnInfo(1,400,50);
    m_pTastView = (CTastListView*)m_wndSplitter.GetPane(1,0);  //获取创建的任务列表
    m_pFtpView = (CFTPView*)m_ChildSplitter.GetPane(0,1);      //获取创建的 FTP 服务器信息窗口
    m_pLocalView = (CLocalView*)m_ChildSplitter.GetPane(0,0);  //获取创建的本地信息窗口
    return TRUE;
}
```

7.5.3 主窗口实现过程

FTP 文件传输管理软件主窗口的实现过程如下：

（1）在创建主窗口时，创建登录信息栏和工具栏窗口。代码如下：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))                  //创建状态栏
    {
        TRACE0("Failed to create status bar\n");
        return -1;
    }
    if (!m_ConnectBar.Create(this, IDD_CONNECTFTPDLG, CBRS_TOP |
        CBRS_TOOLTIPS | CBRS_FLYBY|CBRS_SIZE_DYNAMIC, ID_CONNECTBAR))
    {                                                        //创建登录信息栏
        TRACE0("Failed to create Dialog bar\n");
        return -1;
    }
    m_ConnectBar.EnableDocking(CBRS_ALIGN_TOP | CBRS_ALIGN_BOTTOM);
        EnableDocking(CBRS_ALIGN_ANY);                       //激活登录信息栏的拖放功能
    DockControlBar(&m_ConnectBar);                           //拖入登录信息栏
    //设置登录信息栏编辑框文本
    m_ConnectBar.SetDlgItemText(IDC_FTPSERVER,"192.168.1.23");
    m_ConnectBar.SetDlgItemText(IDC_FTPPORT,"21");
    hMutex = CreateMutex(NULL,false,"mutex1");               //创建一个互斥对象，用于线程同步
    HMODULE hMod = LoadLibrary("dbghelp.dll");               //加载 dbghelp 动态链接库
    MakeSureDirectoryPathExists = (funMakeSure *)GetProcAddress(hMod,
                         "MakeSureDirectoryPathExists");     //获取函数地址
    if (!m_ToolBar.Create(this,IDD_TOOLBAR,CBRS_TOP | CBRS_TOOLTIPS |
```

```
        CBRS_FLYBY|CBRS_SIZE_DYNAMIC,ID_TOOLBARINFO))
    {                                                   //创建工具栏
        TRACE0("Failed to create Dialog bar\n");
        return -1;
    }
    //激活工具栏的拖入功能
    m_ToolBar.EnableDocking(CBRS_ALIGN_TOP | CBRS_ALIGN_BOTTOM);
    EnableDocking(CBRS_ALIGN_ANY);                      //激活框架的拖入功能
    DockControlBar(&m_ToolBar);                         //拖入工具栏
    m_ToolBar.SetDlgItemText(IDC_SAVEDIR,"c:");         //设置工具栏中编辑框文本
    SetIcon(LoadIcon(AfxGetResourceHandle(),MAKEINTRESOURCE(IDI_MAIN)),TRUE);
    return 0;
}
```

（2）处理“查看”菜单中“登录信息栏”菜单项的单击事件，显示或隐藏登录信息栏。代码如下：

```
void CMainFrame::OnLoginbar()
{
    CMenu * pMenu = GetMenu();
    CMenu * pSub = pMenu->GetSubMenu(0);                //获取子菜单
    if (pSub != NULL)
    {
        //获取菜单状态
        int nState = pSub->GetMenuState(ID_CONNECTBAR,MF_BYCOMMAND);
        if (nState == MF_CHECKED)
        {
            //取消菜单的复选标记
            pSub->CheckMenuItem(ID_CONNECTBAR,MF_BYCOMMAND|MF_UNCHECKED);
            CControlBar* pBar = GetControlBar(ID_CONNECTBAR); //获取控制栏
            if (pBar != NULL)                           //隐藏控制栏
            {
                ShowControlBar(pBar, (pBar->GetStyle() & WS_VISIBLE) == 0, FALSE);
            }
            RecalcLayout();                             //界面重新布局
        }
        else if (nState == MF_UNCHECKED)
        {
            //设置菜单复选标记
            pSub->CheckMenuItem(ID_CONNECTBAR,MF_BYCOMMAND|MF_CHECKED);
            CControlBar* pBar = GetControlBar(ID_CONNECTBAR); //获取控制栏
            if (pBar != NULL)                           //显示控制栏
            {
                ShowControlBar(pBar, (pBar->GetStyle() & WS_VISIBLE) == 0, FALSE);
            }
            RecalcLayout();                             //界面重新布局
        }
    }
}
```

（3）处理“查看”菜单中“登录信息栏”菜单项的 UPDATE_COMMAND_UI 消息，根据登录信息栏是否可见设置或取消菜单项的复选标记。代码如下：

```
void CMainFrame::OnUpdateLoginbar(CCmdUI* pCmdUI)
{
	CControlBar* pBar = GetControlBar(pCmdUI->m_nID);	//获取控制栏窗口
	if (pBar != NULL)
	{
		//根据登录信息栏是否可见设置或取消菜单复选标记
		pCmdUI->SetCheck((pBar->GetStyle() & WS_VISIBLE) != 0);
		return;
	}
	pCmdUI->ContinueRouting();	//继续传递消息
}
```

（4）处理“查看”菜单中“工具信息栏”菜单项的单击事件，显示或隐藏工具信息栏。代码如下：

```
void CMainFrame::OnSetToolBar()
{
	CMenu * pMenu = GetMenu();
	CMenu * pSub = pMenu->GetSubMenu(0);	//获取子菜单
	if (pSub != NULL)
	{
		//获取菜单状态
		int nState = pSub->GetMenuState(ID_TOOLBARINFO,MF_BYCOMMAND);
		if (nState == MF_CHECKED)	//判断菜单是否有复选标记
		{
			//取消菜单项的复选标记
			pSub->CheckMenuItem(ID_TOOLBARINFO,MF_BYCOMMAND|MF_UNCHECKED);
			CControlBar* pBar = GetControlBar(ID_TOOLBARINFO); //获取控制条
			if (pBar != NULL)	//隐藏工具栏
			{
				ShowControlBar(pBar, (pBar->GetStyle() & WS_VISIBLE) == 0, FALSE);
			}
			RecalcLayout();	//重新进行窗口布局
		}
		else if (nState == MF_UNCHECKED)
		{
			//设置菜单项的复选标记
			pSub->CheckMenuItem(ID_TOOLBARINFO,MF_BYCOMMAND|MF_CHECKED);
			CControlBar* pBar = GetControlBar(ID_TOOLBARINFO); //获取控制栏
			if (pBar != NULL)
			{
				//显示工具栏
				ShowControlBar(pBar, (pBar->GetStyle() & WS_VISIBLE) == 0, FALSE);
			}
```

```
                RecalcLayout();                                     //重新进行窗口布局
            }
        }
}
```



7.6　登录信息栏设计

7.6.1　登录信息概述

登录信息栏的功能非常简单，就是根据用户输入的 FTP 服务器地址、端口号、用户名和密码登录 FTP 服务器。登录信息栏效果如图 7.12 所示。



图 7.12　登录信息栏

7.6.2　登录界面布局

登录信息栏界面布局如下：

（1）创建一个对话框资源，资源 ID 为 IDD_CONNECTFTPDLG。

（2）向对话框中添加静态文本、编辑框和按钮等控件。

（3）设置主要控件属性，如表 7.1 所示。

表 7.1　登录信息栏窗口控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDD_CONNECTFTPDLG	Style：Child Border：None	CDialogBar：m_ConnectBar
IDC_PASSWORD	Password：TRUE	无
IDC_LOGIN_FTP	Caption：快速登录	无

7.6.3　登录实现过程

登录信息栏实现过程如下：

（1）在主框架类 CMainFrame 中定义一个成员变量 m_ConnectBar，类型为 CDialogBar。

```
CDialogBar   m_ConnectBar;
```

（2）在主框架类 CMainFrame 的 OnCreate 方法中创建登录信息栏，设置登录信息栏中编辑框显示



的默认文本。

```
if (!m_ConnectBar.Create(this, IDD_CONNECTFTPDLG, CBRS_TOP |
    CBRS_TOOLTIPS | CBRS_FLYBY|CBRS_SIZE_DYNAMIC, ID_CONNECTBAR))
{                                                               //创建登录信息栏
    TRACE0("Failed to create Dialog bar\n");
    return -1;
}
    //激活登录信息栏的拖放功能
m_ConnectBar.EnableDocking(CBRS_ALIGN_TOP | CBRS_ALIGN_BOTTOM);
EnableDocking(CBRS_ALIGN_ANY);                                  //激活框架的拖动功能
DockControlBar(&m_ConnectBar);                                  //拖入登录信息栏
//设置登录信息栏编辑框文本
m_ConnectBar.SetDlgItemText(IDC_FTPSERVER,"192.168.1.23");
m_ConnectBar.SetDlgItemText(IDC_FTPPORT,"21");
```

（3）在主框架类 CMainFrame 中添加 LoginFTP 方法实现登录 FTP 服务器的功能。

```
void CMainFrame::LoginFTP()
{
    CString csServer,csPassword,csUser,csPort;
    m_ConnectBar.GetDlgItemText(IDC_FTPSERVER,csServer);        //获取 FTP 服务器 IP
    m_ConnectBar.GetDlgItemText(IDC_FTPPORT,csPort);            //获取 FTP 服务器端口
    m_ConnectBar.GetDlgItemText(IDC_PASSWORD,csPassword);       //获取登录密码
    m_ConnectBar.GetDlgItemText(IDC_USER,csUser);               //获取用户名
    if (!csServer.IsEmpty() && !csPassword.IsEmpty() &&
        !csUser.IsEmpty() && !csPort.IsEmpty())                 //判断登录信息是否为空
    {
        try
        {
            //开始登录服务器
            m_pFtp =  Session.GetFtpConnection(csServer,csUser,csPassword,atoi(csPort));
            m_bLoginSucc = TRUE;
        }
        catch(CInternetException &e)                            //进行异常捕捉
        {
            m_bLoginSucc = FALSE;                               //登录失败
            delete m_pFtp;
        }
        delete m_pFtp;                                          //释放 FTP 连接对象
    }
}
```

（4）在主框架类 CMainFrame 的消息映射部分添加消息映射宏，将“快速登录”按钮的单击事件与 LoginFTP 方法关联。

```
ON_COMMAND(IDC_LOGIN_FTP, LoginFTP)
```



7.7 工具栏窗口设计

7.7.1 工具栏窗口概述

FTP 文件传输管理软件的工具栏窗口实际是一个对话栏（CDialogBar），而不是普通的工具栏（CToolBar），是由于需要在工具栏中设置静态文本和编辑框控件。工具栏窗口效果如图 7.13 所示。

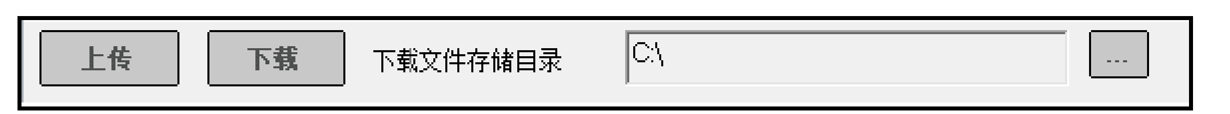


图 7.13 工具栏窗口

7.7.2 工具栏窗口界面布局

工具栏窗口界面布局如下：

（1）创建一个对话框资源，设置资源 ID 为 IDD_TOOLBAR。

（2）向对话框中添加按钮、静态文本和编辑框控件。

（3）设置主要控件属性，如表 7.2 所示。

表 7.2 工具栏窗口控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDD_TOOLBAR	Style：Child Border：None	CDialogBar：m_ToolBar
IDC_BT_UPLOAD	Caption：上传	无
IDC_SAVEDIR	Readonly：TRUE	无
IDC_SETDIR	Caption：…	无

7.7.3 工具栏窗口实现过程

工具栏窗口实现过程如下：

（1）在主框架类 CMainFrame 中定义一个成员变量 m_ToolBar，类型为 CDialogBar。

```
CDialogBar    m_ToolBar;
```

（2）在主框架类 CMainFrame 的 OnCreate 方法中创建工具栏，设置工具栏中的编辑框文本。

```
if (!m_ToolBar.Create(this,IDD_TOOLBAR,CBRS_TOP | CBRS_TOOLTIPS |
     CBRS_FLYBY|CBRS_SIZE_DYNAMIC,ID_TOOLBARINFO))
{                                                              //创建工具栏
     TRACE0("Failed to create Dialog bar\n");
     return -1;
```



```
}
    //激活工具栏的拖放功能
m_ToolBar.EnableDocking(CBRS_ALIGN_TOP | CBRS_ALIGN_BOTTOM);
EnableDocking(CBRS_ALIGN_ANY);                              //激活框架的拖放功能
DockControlBar(&m_ToolBar);                                 //拖入工具栏
m_ToolBar.SetDlgItemText(IDC_SAVEDIR,"c:");                 //设置工具栏编辑框文本
```

（3）向主框架类 CMainFrame 中添加 SetDownLoad 方法，用于设置工具栏中的下载文件存储目录。

```
void CMainFrame::SetDownLoad()
{
    BROWSEINFO BrowInfo;                                    //定义文件浏览对象
    char csFolder[MAX_PATH] = {0};
    memset(&BrowInfo,0,sizeof(BROWSEINFO));
    BrowInfo.hwndOwner = m_hWnd;                            //设置窗口拥有者
    BrowInfo.pszDisplayName = csFolder;                     //记录文件名
    BrowInfo.lpszTitle = "请选择文件";                        //设置浏览对话框标题
    BrowInfo.ulFlags = BIF_EDITBOX;                         //设置浏览对话框标记
    ITEMIDLIST *pitem = SHBrowseForFolder(&BrowInfo);       //弹出浏览对话框
    if (pitem)
    {
        SHGetPathFromIDList(pitem,csFolder);                //获取目录
        m_csDownDir = csFolder;
        m_ToolBar.SetDlgItemText(IDC_SAVEDIR,m_csDownDir);
    }
}
```

（4）在主框架类 CMainFrame 的消息映射部分添加消息映射宏，将“…”按钮的单击事件与 SetDownLoad 方法关联。

```
ON_COMMAND(IDC_SETDIR,SetDownLoad)
```

（5）向主框架类 CMainFrame 中添加 OnDownload 方法，实现 FTP 目录或文件的下载。

```
void CMainFrame::OnDownload()
{
    if (m_bLoginSucc && m_pLocalView != NULL)
    {
        m_ToolBar.GetDlgItemText(IDC_SAVEDIR,m_csDownDir);        //获取本地存储根目录
        POSITION pos = m_pFtpView->m_RemoteFiles.GetFirstSelectedItemPosition();
        int nSelCount = m_pFtpView->m_RemoteFiles.GetSelectedCount(); //获取下载的任务数量
        while (pos != NULL)                                        //利用循环遍历文件或目录
        {
            int nItem = m_pFtpView->m_RemoteFiles.GetNextSelectedItem(pos);
            LVITEM item;
            item.mask = LVIF_TEXT;
            item.cchTextMax = MAX_PATH;
            char text[MAX_PATH] = {0};
            item.pszText = text;
```

```
                item.iItem = nItem;
                item.iSubItem = 0;
                m_pFtpView->m_RemoteFiles.GetItem(&item);              //获取视图项信息
                int nCount = m_pTastView->m_TastList.GetItemCount();
                int nCurItem;
                //向任务列表中插入一个行数据，表示当前任务
                nCurItem = m_pTastView->m_TastList.InsertItem(nCount,"");
                m_pTastView->m_TastList.SetItemData(nCurItem,nCurItem);
                m_pTastView->m_TastList.SetItemText(nCurItem,0,text);
                m_pTastView->m_TastList.SetItemText(nCurItem,1,"下载");
                m_pTastView->m_TastList.SetItemText(nCurItem,2,m_csServer);
                m_pTastView->m_TastList.SetItemText(nCurItem,3,"正在下载");
                ThreadParam * Param = new ThreadParam();               //创建线程参数
                Param->pDlg = this;                                     //填充线程参数
                Param->nItem = nCurItem;
                Param->nDownFlag =m_pFtpView->m_RemoteFiles.GetItemData(nItem);
                memset(Param->m_DownFile,0,MAX_PATH);
                memset(Param->m_RelativeFile,0,MAX_PATH);
                strcpy(Param->m_DownFile, m_pFtpView->m_RemoteFiles.m_CurDir);
                strcat(Param->m_DownFile,text);
                strcpy(Param->m_RelativeFile,text);
                //创建新的线程，执行线程函数
                HANDLE hHandle = CreateThread(NULL,0,DownloadThreadProc,(void*)Param,0,NULL);
                m_pTastView->m_TastList.SetItemData(nCurItem,(DWORD)hHandle);
            }
        }
}
```

（6）在主框架类 CMainFrame 的消息映射部分添加消息映射宏，将“下载”按钮的单击事件与 OnDownload 方法关联。

```
ON_COMMAND(IDC_BT_DOWNLOAD,OnDownload)
```

（7）定义全局的线程函数，在下载文件时将在线程函数中调用下载函数实现文件下载。

```
//下载文件的线程函数
DWORD __stdcall DownloadThreadProc(LPVOID lpParameter)
{
        ThreadParam *Param = (ThreadParam *)lpParameter;           //获取线程参数
        CMainFrame * pDlg = Param->pDlg;                           //获取主对话框指针
        int nItem = Param->nItem;                                  //获取当前下载任务对应的视图项索引
        char downfile[MAX_PATH] = {0};
        strcpy(downfile,Param->m_DownFile);                       //复制下载文件名
        char relfile[MAX_PATH] = {0};
        strcpy(relfile,Param->m_RelativeFile);                    //复制相对路径
        pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"正在下载");
        if (Param->nDownFlag ==0)                                  //当前选择的是文件
        {
                //调用 DownLoadFile 方法下载文件
```



```
            pDlg->DownLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,true,pDlg->m_csServer,
                    pDlg->m_csUser,pDlg->m_csPassword,pDlg->m_nPort,pDlg->m_csDownDir);
        }
        else if(Param->nDownFlag ==1)
        {
            //调用 DownLoadFile 方法下载文件

        pDlg->DownLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,false,pDlg->m_csServepDlg->
                                        m_csUser,pDlg->m_csPassword,pDlg->m_nPort,pDlg->m_csDownDir);
        }
        pDlg->m_pTastView->m_TastList.SetItemText(nItem,3,"完成");
        pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"下载完成");
        if (pDlg->m_dwStop == (DWORD)Param->m_hThread)          //终止线程后设置初始标记
        {
            pDlg->m_dwStop = 0;                                  //恢复任务终止标记
        }
        //任务完成后删除任务列表中对应的视图项
        pDlg->DeleteItemFormData(&pDlg->m_pTastView->m_TastList,(DWORD)Param->m_hThread);
        int nCount = pDlg->m_pTastView->m_TastList.GetItemCount();
        delete Param;                                            //释放线程参数
        if (pDlg->m_bTurnOff && nCount == 0)                     //下载后是否关机
        {
            pDlg->TurnOff();                                     //关机操作
        }
        return 0;
}
```

（8）向主框架类 CMainFrame 中添加 OnUpload 方法，实现文件的上传。文件上传与下载的思路是相同的，均是通过单独的线程来完成的。

```
void CMainFrame::OnUpload()
{
        if (m_bLoginSucc && m_pLocalView != NULL)
        {
            //获取本地列表视图中选项的位置
            POSITION pos = m_pLocalView->m_LocalFiles.GetFirstSelectedItemPosition();
            //获取本地列表视图中选项数量
            int nSelCount = m_pLocalView->m_LocalFiles.GetSelectedCount();
            while (pos != NULL)
            {
                //获取视图项索引
                int nItem = m_pLocalView->m_LocalFiles.GetNextSelectedItem(pos);
                LVITEM item;
                item.mask = LVIF_TEXT;
                item.cchTextMax = MAX_PATH;
                char text[MAX_PATH] = {0};
                item.pszText = text;
                item.iItem = nItem;
                item.iSubItem = 0;
```

```
                m_pLocalView->m_LocalFiles.GetItem(&item);              //获取视图项信息
                int nCount = m_pTastView->m_TastList.GetItemCount();
                int nCurItem;
                //向任务列表中添加上传文件的任务信息
                nCurItem = m_pTastView->m_TastList.InsertItem(nCount,"");
                m_pTastView->m_TastList.SetItemData(nCurItem,nCurItem);
                m_pTastView->m_TastList.SetItemText(nCurItem,0,text);
                m_pTastView->m_TastList.SetItemText(nCurItem,1,"上传");
                m_pTastView->m_TastList.SetItemText(nCurItem,2,m_csServer);
                m_pTastView->m_TastList.SetItemText(nCurItem,3,"正在上传");
                ThreadParam * Param = new ThreadParam();                //定义线程参数
                Param->pDlg = this;                                     //填充线程参数
                Param->nItem = nCurItem;
                Param->nDownFlag = m_pLocalView->m_LocalFiles.GetItemData(nItem);
                memset(Param->m_DownFile,0,MAX_PATH);
                memset(Param->m_RelativeFile,0,MAX_PATH);
                strcpy(Param->m_DownFile, m_pLocalView->m_LocalFiles.m_CurDir);
                strcat(Param->m_DownFile,text);                         //设置下载文件
                CString csRelative   = m_pFtpView->m_RemoteFiles.m_BaseDir;
                if (csRelative.Right(1) != "/" && csRelative.GetLength()>1)
                    csRelative += "/";
                csRelative +=   m_pFtpView->m_RemoteFiles.m_CurDir;
                if (csRelative.Right(1) != "/" && csRelative.GetLength()>1)
                    csRelative += "/";
                csRelative += text;
                strcpy(Param->m_RelativeFile,csRelative);               //设置相对路径
                //创建一个线程，执行文件上传任务
                HANDLE hHandle = CreateThread(NULL,0,UploadThreadProc,(void*)Param,0,NULL);
                m_pTastView->m_TastList.SetItemData(nCurItem,(DWORD)hHandle);
                Param->m_hThread = hHandle;
            }
        }
}
```

（9）向主框架类 CMainFrame 中添加消息映射宏，将“上传”按钮的单击事件与 OnUpload 方法关联。

```
ON_COMMAND(IDC_BT_UPLOAD,OnUpload)
```

（10）添加全局线程函数 UploadThreadProc，在上传文件时将创建一个线程，在线程函数中调用上传函数实现文件上传。

```
//上传文件的线程函数
DWORD __stdcall UploadThreadProc(LPVOID lpParameter)
{
    ThreadParam *Param = (ThreadParam *)lpParameter;                //获取线程参数
    CMainFrame * pDlg = Param->pDlg;                                //获取主对话框指针
    int nItem = Param->nItem;
```



```
    char downfile[MAX_PATH] = {0};
    strcpy(downfile,Param->m_DownFile);                     //获取下载文件
    char relfile[MAX_PATH] = {0};
    strcpy(relfile,Param->m_RelativeFile);                  //获取相对路径
    pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"正在上传");
    if (Param->nDownFlag ==0)                               //当前选择的是文件
    {
        //调用 UpLoadFile 方法上传文件
        pDlg->UpLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,true,pDlg->m_csServer,
                    pDlg->m_csUser,pDlg->m_csPassword,pDlg->m_nPort);
    }
    else if(Param->nDownFlag ==1)                           //当前选择的是目录
    {
        //调用 UpLoadFile 方法上传文件
        pDlg->UpLoadFile(pDlg,downfile,relfile,(DWORD)Param->m_hThread,false,pDlg->m_csServer,
                    pDlg->m_csUser,pDlg->m_csPassword,pDlg->m_nPort);
    }

    if (pDlg->m_dwStop == (DWORD)Param->m_hThread)          //终止线程后设置初始标记
    {
        pDlg->m_dwStop = 0;                                 //恢复线程终止标记
    }

    pDlg->m_pTastView->m_TastList.SetItemText(nItem,3,"完成");
    pDlg->m_pTastView->m_TastList.SetItemText(nItem,4,"上传完成");
    //在文件上传任务完成后从任务列表中删除指定的任务信息
    pDlg->DeleteItemFormData(&pDlg->m_pTastView->m_TastList,(DWORD)Param->m_hThread);
    delete Param;                                           //释放线程参数
    int nCount = pDlg->m_pTastView->m_TastList.GetItemCount();
    if (pDlg->m_bTurnOff && nCount==0)                      //是否为上传后关机
    {
        pDlg->TurnOff();                                    //关机操作
    }
    return 0;
}
```

详细代码请见资源包中的源码。

7.8　本地信息窗口设计


视频讲解

7.8.1　本地信息窗口概述

本地信息窗口的主要作用是加载本地系统目录和文件到列表视图控件中，并实现通过鼠标拖曳上传文件。本地信息窗口效果如图 7.14 所示。

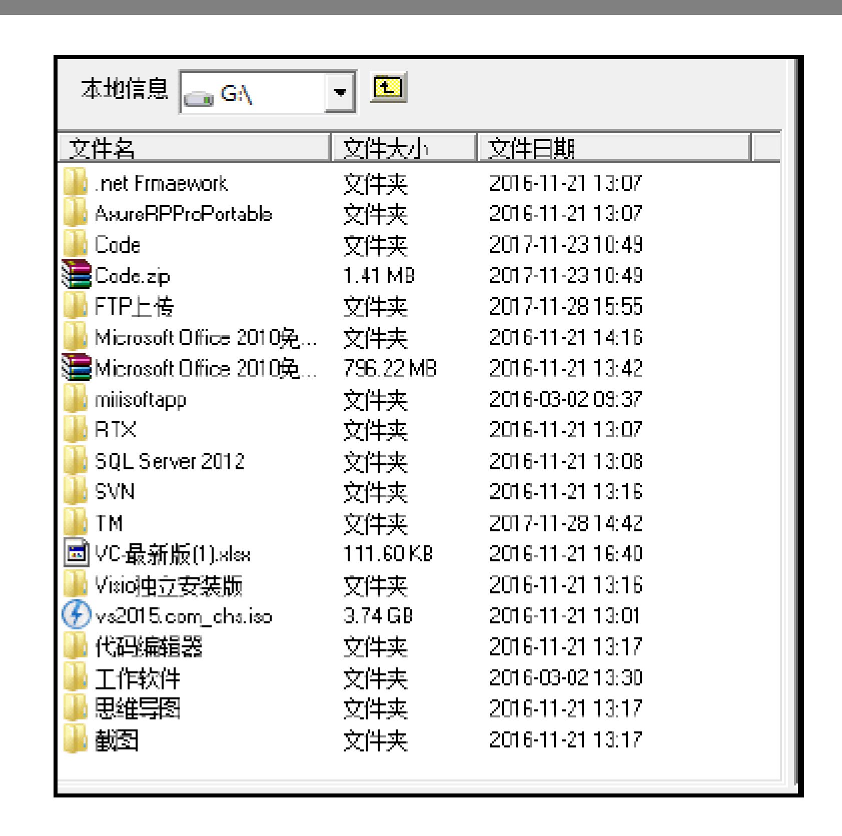


图 7.14　本地信息窗口

7.8.2　本地信息窗口界面布局

本地信息窗口设计过程如下：

（1）新建一个窗体视图类，类名为 CLocalView，基类为 CFormView。

（2）向对话框中添加按钮、静态文本、图片、列表视图等控件。

（3）设置控件属性，如表 7.3 所示。

表 7.3　本地信息窗口控件属性设置

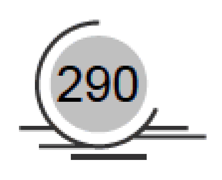
控件 ID	控 件 属 性	关 联 变 量
IDD_LOCALVIEW_FORM	Style：Child Border：None	CLocalView：m_pLocalView
IDC_FRAME	Type：Frame Color：Black	CStatic：m_Frame
IDC_BACK	Type：Bitmap Image：IDB_BACKBTN	无
IDC_LOCALFILES	View：Report	CSortListCtrl：m_LocalFiles

7.8.3　本地信息窗口实现过程

本地信息窗口实现过程如下：

（1）向窗体视图类中添加 GetSysDisk 方法，利用 GetLogicalDriveStrings 函数获取系统磁盘目录，将其显示在组合框中。

```
void CLocalView::GetSysDisk()
{
	m_DiskList.ResetContent();                                        //删除组合框中的数据
	char	pchDrives[128] = {0};
	char* pchDrive;
```



```
	GetLogicalDriveStrings(sizeof(pchDrives), pchDrives);		//获取系统磁盘目录
	pchDrive = pchDrives;
	int nItem = 0;
	while(*pchDrive)							//列举系统磁盘目录
	{
		COMBOBOXEXITEM   cbi;					//定义组合框选项信息
		CString   csText;
		cbi.mask = CBEIF_IMAGE | CBEIF_INDENT | CBEIF_OVERLAY |
			CBEIF_SELECTEDIMAGE | CBEIF_TEXT;			//设置组合框选项标记
		SHFILEINFO shInfo;						//定义外壳文件信息对象
		int nIcon;
		//获取文件图标
		SHGetFileInfo(pchDrive,0,&shInfo,sizeof(shInfo),SHGFI_ICON | SHGFI_SMALLICON);
		nIcon = shInfo.iIcon;					//获取图标索引
		cbi.iItem = nItem;						//填充组合框选项信息
		cbi.pszText = pchDrive;
		cbi.cchTextMax = strlen(pchDrive);
		cbi.iImage = nIcon;
		cbi.iSelectedImage = nIcon;
		cbi.iOverlay = 0;
		cbi.iIndent = (0 & 0x03);
		m_DiskList.InsertItem(&cbi);				//向组合框中添加选项信息
		nItem ++;
		pchDrive += strlen(pchDrive) + 1;			//获取下一个磁盘目录
	}
}
```

（2）向窗体视图类中添加 GetSysFileImage 方法，获取系统文件图像列表。

```
void CLocalView::GetSysFileImage()
{
	CFTPManageApp *pApp = (CFTPManageApp*)AfxGetApp();		//获取应用程序对象指针
	m_DiskList.SetImageList(&pApp->m_ImageList);			//设置组合框的图像列表
	m_LocalFiles.SetImageList(&pApp->m_ImageList,LVSIL_SMALL);	//设置列表视图的图像列表
}
```

（3）在窗体视图初始化时创建一个群组框显示系统磁盘目录，修改列表视图控件的扩展风格，向列表视图控件中添加列。

```
void CLocalView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();
	CRect ComBoxRC;
	m_Frame.GetWindowRect(ComBoxRC);					//获取静态文本控件的区域
	m_Frame.ScreenToClient(ComBoxRC);
	m_Frame.MapWindowPoints(this,ComBoxRC);
	ComBoxRC.bottom = ComBoxRC.top+100;
	m_DiskList.Create(WS_CHILD|WS_VISIBLE|CBS_DROPDOWNLIST|
		CBS_SORT,ComBoxRC,this,112);				//创建组合框控件
```

```
    //设置列表视图控件的风格
    m_LocalFiles.SetExtendedStyle(LVS_EX_TWOCLICKACTIVATE|LVS_EX_FULLROWSELECT);
    //设置列表视图控件的列信息
    m_LocalFiles.SetColumns(_T("文件名,150;文件大小,80;文件日期,150"));
    m_bSubClass = TRUE;
    GetSysFileImage();                                  //获取系统文件图像列表
    GetSysDisk();                                       //加载系统磁盘目录
    SIZE szWidth,szHeight;
    szWidth.cx = szWidth.cy = 0;
    szHeight.cx = szHeight.cy = 0;
    SetScrollSizes(MM_TEXT,szWidth,szHeight);
}
```

（4）处理窗体视图的 WM_SIZE 消息，当用户调整窗体视图大小时，适当调整列表视图控件的大小。

```
void CLocalView::OnSize(UINT nType, int cx, int cy)
{
    CFormView::OnSize(nType, cx, cy);
    if (m_bSubClass==TRUE)
    {
        CRect rc;
        GetClientRect(rc);                              //获取视图窗口客户区域
        rc.DeflateRect(0,40,0,0);                       //修改区域的高度
        m_LocalFiles.MoveWindow(rc);                    //设置列表视图控件显示的区域
    }
}
```

（5）处理组合框中选项改变时的事件，根据当前显示的磁盘目录，在列表视图控件中显示相应的子目录和文件。

```
void CLocalView::DiskItemChange()
{
    COMBOBOXEXITEM cbi;                                 //定义组合框项目信息
    memset(&cbi,0,sizeof(COMBOBOXEXITEM));              //初始化项目信息
    int nCurItem = m_DiskList.GetCurSel();              //获取当前选中的项目
    cbi.mask =   CBEIF_TEXT;
    char chName[128] = {0};
    cbi.pszText = chName;
    cbi.cchTextMax = 128;
    cbi.iItem = nCurItem;
    m_DiskList.GetItem(&cbi);                           //获取项目信息
    m_LocalFiles.m_BaseDir = chName;                    //读取项目文本
    m_LocalFiles.DisplayPath(chName);                   //根据当前磁盘目录加载本地磁盘信息
}
```

（6）向窗体视图中添加 OnBack 方法，当用户单击图片控件时将调用该方法返回上一级目录。

```
void CLocalView::OnBack()
{
```



```
    //处理列表视图控件的 WM_KEYDOWN 消息来返回上一级目录
    m_LocalFiles.SendMessage(WM_KEYDOWN,VK_BACK,0);
}
```

（7）处理列表视图控件的 LVN_BEGINDRAG 消息，开始鼠标拖曳。

```
void CLocalView::OnBegindragLocalfiles(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    *pResult = 0;
    int nItem=pNMListView->iItem;                          //获取拖动的视图项索引
    POINT pt=pNMListView->ptAction;                        //获取坐标
    m_pDragList = m_LocalFiles.CreateDragImage(nItem,&pt); //创建拖动的图像列表
    m_pDragList->BeginDrag(0,CPoint(0,0));                 //开始鼠标拖曳
    m_pDragList->DragEnter(NULL,pt);
    SetCapture();                                          //设置鼠标捕捉
    m_bDrag = TRUE;
    *pResult = 0;
}
```

（8）在鼠标移动过程中设置图像拖动的位置。

```
void CLocalView::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_bDrag)
    {
        CRect rect;
        GetWindowRect(&rect);                              //获取窗口区域
        ClientToScreen(&point);                            //转换区域坐标
        m_pDragList->DragMove(point);                      //沿着鼠标拖动图像
    }
    CFormView::OnMouseMove(nFlags, point);
}
```

（9）在释放鼠标按钮时结束拖曳。如果拖放到远程 FTP 服务器信息窗口的列表视图控件中，将进行文件上传操作。

```
void CLocalView::OnLButtonUp(UINT nFlags, CPoint point)
{
    if(m_bDrag)
    {
        m_bDrag = FALSE;                                   //设置鼠标拖动结束标记
        CImageList::DragLeave(NULL);                       //鼠标拖动结束
        CImageList::EndDrag();
        ReleaseCapture();                                  //释放鼠标捕捉
        delete m_pDragList;                                //删除图像列表
        m_pDragList=NULL;
        CRect rect;
        CMainFrame * pDlg = (CMainFrame*)AfxGetMainWnd();  //获取主窗口指针
```

```
            //获取远程 FTP 窗口中的列表视图控件区域
            pDlg->m_pFtpView->m_RemoteFiles.GetWindowRect(&rect);
            ClientToScreen(&point);                           //转换区域坐标
            if(rect.PtInRect(point))                          //判断是否拖放到远程 FTP 窗口的列表视图中
            {
                pDlg->OnUpload();                             //执行上传操作
            }
        }
        CFormView::OnLButtonUp(nFlags, point);
}
```


视频讲解

7.9 远程 FTP 服务器信息窗口设计

7.9.1 远程 FTP 服务器信息窗口概述

远程 FTP 服务器信息窗口的主要作用是显示 FTP 服务器上的文件和目录，并实现通过鼠标拖曳来完成 FTP 文件的下载。远程 FTP 服务器信息窗口效果如图 7.15 所示。

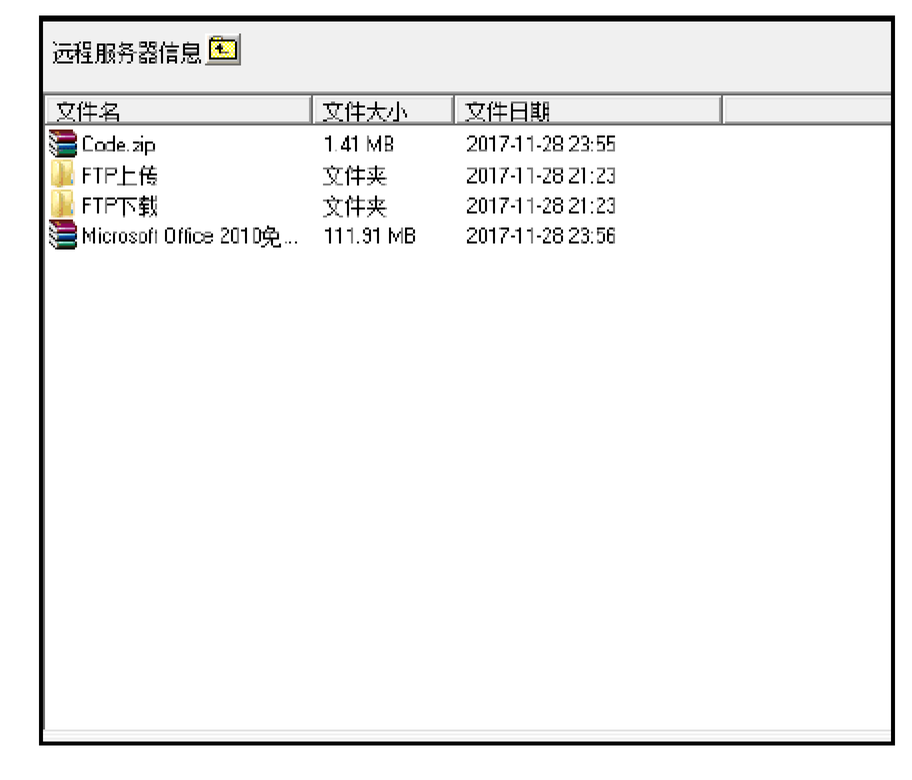


图 7.15　远程 FTP 服务器信息窗口

7.9.2 远程 FTP 服务器信息窗口界面布局

远程 FTP 服务器信息窗口设计过程如下：

（1）新建一个窗体视图类，类名为 CFTPView，基类为 CFormView。

（2）向对话框中添加按钮、静态文本、图片、列表视图等控件。

（3）设置控件属性，如表 7.4 所示。

表 7.4　远程 FTP 服务器信息窗口控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDD_FTPVIEW_FORM	Style：Child Border：None	CFTPView：m_pFtpView
IDC_BACK	Type：Bitmap Image：IDB_BACKBTN	无
IDC_REMOTEFILES	View：Report	CSortListCtrl：m_RemoteFiles

7.9.3 远程 FTP 服务器信息窗口实现过程

远程 FTP 服务器信息窗口实现过程如下：



（1）向窗体视图类中添加 GetSysFileImage 方法，获取系统文件图像列表。

```
void CFTPView::GetSysFileImage()
{
    CFTPManageApp *pApp = (CFTPManageApp*)AfxGetApp();           //获取应用程序指针
    m_RemoteFiles.SetImageList(&pApp->m_ImageList,LVSIL_SMALL);   //设置列表视图的图像列表
}
```

（2）在窗体视图初始化时设置列表视图控件风格、向列表视图控件中添加列，获取系统文件图标。

```
void CFTPView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    //修改列表视图控件的扩展风格
    m_RemoteFiles.SetExtendedStyle(LVS_EX_TWOCLICKACTIVATE|LVS_EX_FULLROWSELECT);
    //设置列表视图的列信息
    m_RemoteFiles.SetColumns(_T("文件名,150;文件大小,80;文件日期,150"));
    m_bSubClass = TRUE;
    GetSysFileImage();                                             //获取系统文件图像列表
    m_RemoteFiles.m_nListType = 1;                                 //显示 FTP 服务器信息
    SIZE szWidth,szHeight;
    szWidth.cx = szWidth.cy = 0;
    szHeight.cx = szHeight.cy = 0;
    SetScrollSizes(MM_TEXT,szWidth,szHeight);
    SendMessage(WM_SIZE,0,0);
}
```

（3）处理窗体视图的 WM_SIZE 消息，当用户调整窗体视图大小时，适当调整列表视图控件的大小。

```
void CFTPView::OnSize(UINT nType, int cx, int cy)
{
    CFormView::OnSize(nType, cx, cy);
    if (m_bSubClass==TRUE)
    {
        CRect rc;
        GetClientRect(rc);                                         //获取窗口客户区域
        rc.DeflateRect(0,40,0,0);                                  //调整区域大小
        m_RemoteFiles.MoveWindow(rc);                              //调整列表视图控件区域
    }
}
```

（4）向窗体视图中添加 OnBack 方法，当用户单击图片控件时将调用该方法返回上一级目录。

```
void CFTPView::OnBack()
{
    //触发列表视图控件的 WM_KEYDOWN 消息，返回上一级目录
    m_RemoteFiles.SendMessage(WM_KEYDOWN,VK_BACK,0);
}
```

（5）处理列表视图控件的 LVN_BEGINDRAG 消息，开始鼠标拖曳。

```
void CFTPView::OnBegindragRemotefiles(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
	*pResult = 0;
	int nItem=pNMListView->iItem;                               //获取拖动的视图项
	POINT pt=pNMListView->ptAction;                             //获取当前坐标
	m_pDragList = m_RemoteFiles.CreateDragImage(nItem,&pt);     //创建一个拖动图像列表
	m_pDragList->BeginDrag(0,CPoint(0,0));                      //开始拖动
	m_pDragList->DragEnter(NULL,pt);
	SetCapture();                                               //捕捉鼠标
	m_bDrag = TRUE;                                             //设置拖动标记
	*pResult = 0;
}
```

（6）在鼠标移动过程中设置图像拖动的位置。

```
void CFTPView::OnMouseMove(UINT nFlags, CPoint point)
{
	if (m_bDrag)                                                //拖动过程中
	{
		CRect rect;
		GetWindowRect(&rect);                                   //获取窗口区域
		ClientToScreen(&point);                                 //转换坐标
		m_pDragList->DragMove(point);                           //沿着鼠标拖动图像
	}
	CFormView::OnMouseMove(nFlags, point);
}
```

（7）在释放鼠标按钮时结束拖曳。如果拖放到本地信息窗口的列表视图控件中，将进行文件下载操作。

```
void CFTPView::OnLButtonUp(UINT nFlags, CPoint point)
{
	if(m_bDrag)
	{
		m_bDrag = FALSE;                                        //设置拖动结束标记
		CImageList::DragLeave(NULL);                            //释放鼠标拖动
		CImageList::EndDrag();
		ReleaseCapture();                                       //释放鼠标捕捉
		delete m_pDragList;                                     //释放拖动的图像列表
		m_pDragList=NULL;
		CRect rect;
		CMainFrame * pDlg = (CMainFrame*)AfxGetMainWnd();
		pDlg->m_pLocalView->m_LocalFiles.GetWindowRect(&rect);
		ClientToScreen(&point);                                 //转换客户坐标
		if(rect.PtInRect(point))                                //判断是否拖放到本地列表视图中
		{
```



```
                pDlg->OnDownload();                              //执行下载操作
            }
        }
        CFormView::OnLButtonUp(nFlags, point);
}
```

7.10　任务列表窗口设计



7.10.1　任务列表窗口概述

任务列表窗口主要用于显示当前的上传、下载任务，用户可以暂停或取消某一任务。任务列表窗口如图 7.16 所示。

图 7.16　任务列表窗口

7.10.2　任务列表窗口界面布局

任务列表窗口界面布局如下：

（1）新建一个窗体视图类，类名为 CTastListView，基类为 CFormView。

（2）向对话框中添加按钮和列表视图等控件。

（3）设置控件属性，如表 7.5 所示。

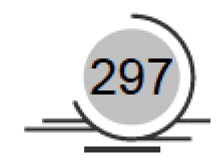
表 7.5　任务列表窗口控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDD_TASTLISTVIEW_FORM	Style：Child Border：None	CtastListView：m_pTastView
IDC_BT_STOP	Caption：暂停	无
IDC_TASKLIST	View：Report	CListCtrl：m_TastList

7.10.3　任务列表窗口实现过程

任务列表窗口实现过程如下：

（1）在窗体视图初始化时设置列表视图控件风格，向列表视图控件中添加列，调整窗口中按钮的位置。



```
void CTastListView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();
	//设置列表视图控件的扩展风格
	m_TastList.SetExtendedStyle(LVS_EX_TWOCLICKACTIVATE|LVS_EX_FULLROWSELECT|
		LVS_EX_GRIDLINES);
	//向列表视图控件中添加列
	m_TastList.InsertColumn(0,"任务名称",LVCFMT_LEFT,120);
	m_TastList.InsertColumn(1,"任务类型",LVCFMT_LEFT,80);
	m_TastList.InsertColumn(2,"主机信息",LVCFMT_LEFT,80);
	m_TastList.InsertColumn(3,"状态",LVCFMT_LEFT,120);

	CRect rc;
	GetClientRect(rc);                                  //获取窗口客户区域
	rc.DeflateRect(0,0,100,0);                          //调整区域宽度
	m_TastList.MoveWindow(rc);                          //设置列表视图显示的区域
	CRect btnRC;                                        //按钮显示区域
	btnRC.left = rc.right + 10;
	btnRC.right = btnRC.left + 80;
	btnRC.top = rc.top + 20;
	btnRC.bottom = btnRC.top + 20;
	btnRC.OffsetRect(0,30);
	m_Continue.MoveWindow(btnRC);                       //设置“继续”按钮的显示区域
	btnRC.OffsetRect(0,30);
	m_Stop.MoveWindow(btnRC);                           //设置“停止”按钮的显示区域
	btnRC.OffsetRect(0,30);
	m_Delete.MoveWindow(btnRC);                         //设置“删除”按钮的显示区域
	m_bSubClass = TRUE;
	SIZE szWidth,szHeight;
	szWidth.cx = szWidth.cy = 0;
	szHeight.cx = szHeight.cy = 0;
	SetScrollSizes(MM_TEXT,szWidth,szHeight);
}
```

（2）处理窗体视图的 WM_SIZE 消息，在窗体视图大小改变时，调整列表视图控件的大小以及按钮的位置。

```
void CTastListView::OnSize(UINT nType, int cx, int cy)
{
	CFormView::OnSize(nType, cx, cy);
	if (m_bSubClass==TRUE)
	{
		CRect rc;
		GetClientRect(rc);                              //获取视图窗口客户区域
		rc.DeflateRect(0,0,100,0);                      //调整区域宽度
		m_TastList.MoveWindow(rc);                      //设置列表视图控件的显示区域
		CRect btnRC;                                    //按钮显示区域
		btnRC.left = rc.right + 10;
		btnRC.right = btnRC.left + 80;
```



```
        btnRC.top = rc.top + 20;
        btnRC.bottom = btnRC.top + 20;
        btnRC.OffsetRect(0,10);
        m_Stop.MoveWindow(btnRC);                        //设置“暂停”按钮显示区域
        btnRC.OffsetRect(0,30);
        m_Continue.MoveWindow(btnRC);                    //设置“继续”按钮显示区域
        btnRC.OffsetRect(0,30);
        m_Delete.MoveWindow(btnRC);                      //设置“删除”按钮显示区域
    }
}
```

（3）处理“暂停”按钮的单击事件，暂停某一个任务的执行。

```
void CTastListView::OnBtStop()
{
    int nSel = m_TastList.GetSelectionMark();                    //获取当前选中的项目
    if (nSel != -1)
    {
        DWORD nItemData = m_TastList.GetItemData(nSel);          //获取与任务关联的线程句柄
        CString csState = m_TastList.GetItemText(nSel,3);
        if (csState != "暂停")
        {
            SuspendThread((HANDLE)nItemData);                    //挂起线程
            m_TastList.SetItemText(nSel,3,"暂停");               //设置任务状态
        }
    }
}
```

（4）处理“继续”按钮的单击事件，恢复某一个暂停的任务。

```
void CTastListView::OnBtContinue()
{
    int nSel = m_TastList.GetSelectionMark();                    //获取当前选中的选项
    if (nSel != -1)
    {
        DWORD nItemData = m_TastList.GetItemData(nSel);          //获取任务对应的线程句柄
        CString csType = m_TastList.GetItemText(nSel,1);
        ResumeThread((HANDLE)nItemData);                         //唤醒线程
        if (csType=="下载")
        {
            m_TastList.SetItemText(nSel,3,"正在下载");           //设置任务的当前状态
        }
        else
        {
            m_TastList.SetItemText(nSel,3,"正在上传");           //设置任务的当前状态
        }
    }
}
```

（5）处理“删除”按钮的单击事件，将取消某一任务的执行。

```
void CTastListView::OnBtDelete()
{
    int nSel = m_TastList.GetSelectionMark();                           //获取当前选择的项目
    if (nSel != -1)
    {
        DWORD nItemData = m_TastList.GetItemData(nSel);                 //获取当前任务的线程句柄
        CMainFrame * pDlg = (CMainFrame*)AfxGetMainWnd();               //获取主窗口指针
        pDlg->m_dwStop = nItemData;                                     //设置任务结束标记为当前的线程句柄
    }
}
```

7.11 项目文件清单

FTP 管理系统项目文件清单如表 7.6 所示。

表 7.6 项目文件清单

文 件 名 称	文 件 类 型	文 件 描 述	文 件 名 称	文 件 类 型	文 件 描 述
FTPManage.cpp	源文件	FTP 管理	MainFrm.cpp	源文件	主架构
FTPManage.rc	资源文件	FTP 管理资源文件	SortHeaderCtrl.cpp	源文件	分类控制
FTPManageDoc.cpp	源文件	FTP 管理文件	SortListCtrl.cpp	源文件	分类列表控制
FTPManageView.cpp	源文件	FTP 管理视图	TaskListView.cpp	源文件	任务列表
FTPView.cpp	源文件	远程 FTP 服务器	LocalView.cpp	源文件	本地信息

7.12 本章总结

FTP 管理系统是通过 Visual Studio 2017 实现的，实现的功能是使局域网内任意计算机都能多任务上传和下载文件。通过本章的学习，希望读者能够了解开发的流程，以及有关 TCP/IP 协议的知识，并能够将本章的系统应用在生活中。

第 8 章

媒体播放器

（Visual Studio 2017+Direct Show 实现）

在互联网上有许多优秀的媒体播放器软件，功能也各具千秋，例如金山影霸、影音风暴、Windows 的 MediaPlayer 等。本章将要设计一个媒体播放器，特色功能就是实现字幕叠加，以及图像的亮度、饱和度、对比度的设置。

通过学习本章，读者可以学到：

- 视频图像的字幕叠加
- 视频图像的亮度、饱和度、对比度设置
- 视频图像的黑白效果显示


配置说明



8.1 开发背景

随着社会的发展，越来越多的人开始接触计算机，也有越来越多的人喜欢在上网、工作时，播放一些音乐，适当娱乐一下，缓解一下紧张情绪。所以，应运而生的是音频播放器软件，能够为用户播放常见格式的音频文件。并且，在使用的过程中，也尽可能使设计更人性化。比如，能最小化到托盘并可以进行操作。良好的人机交互界面，也能给人以美好的感官享受。

8.2 需求分析

目前，市场上已经有很多音乐播放器，因此本次的设计不仅可以控制播放进度、音量大小、全屏切换等基本功能，还具备以下特点。

☑ 能够字幕叠加。
☑ 能够控制图像颜色。
☑ 用户可以通过设计自己的过滤器实现以上两个特色功能。

8.3 系统设计

8.3.1 系统目标

音乐播放器这个系统能够方便用户使用，操作起来灵活，视觉上美观。设计本系统时应该完成以下几个目标。

☑ 播放器的界面美观。
☑ 在屏幕上有固定的格式。
☑ 能够写入计算机磁盘音乐。
☑ 运行稳定、安全可靠。

8.3.2 系统功能结构

系统功能结构如图 8.1 所示。

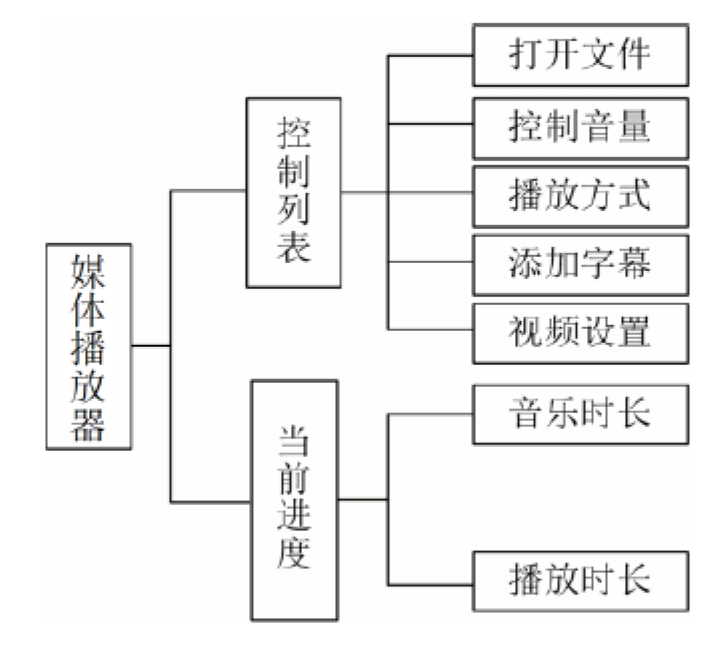


图 8.1 媒体播放器系统功能图

8.3.3 系统 览

媒体播放器主要包括媒体播放器、视频显示、文件播放列表、字幕叠加和视频设置 5 个窗口，下面分别给出各个窗口的效果图。





媒体播放器主窗口如图 8.2 所示。在该模块中，能够显示媒体文件的播放进度、播放时间等，并能控制播放进度、音量大小以及调用字幕叠加、视频设置等子窗口。

视频显示窗口用于显示视频图像画面，效果如图 8.3 所示。



图 8.2　媒体播放器主窗口



图 8.3　视频显示窗口

文件播放列表窗口如图 8.4 所示。该模块主要用于播放一组媒体文件，用户可以在磁盘中选择要加载的媒体文件，同时可以设置一组媒体文件循环播放或随机播放。

字幕叠加窗口如图 8.5 所示。该模块主要用于字幕的添加与删除，用户可以设置显示字幕的内容、字体和位置。

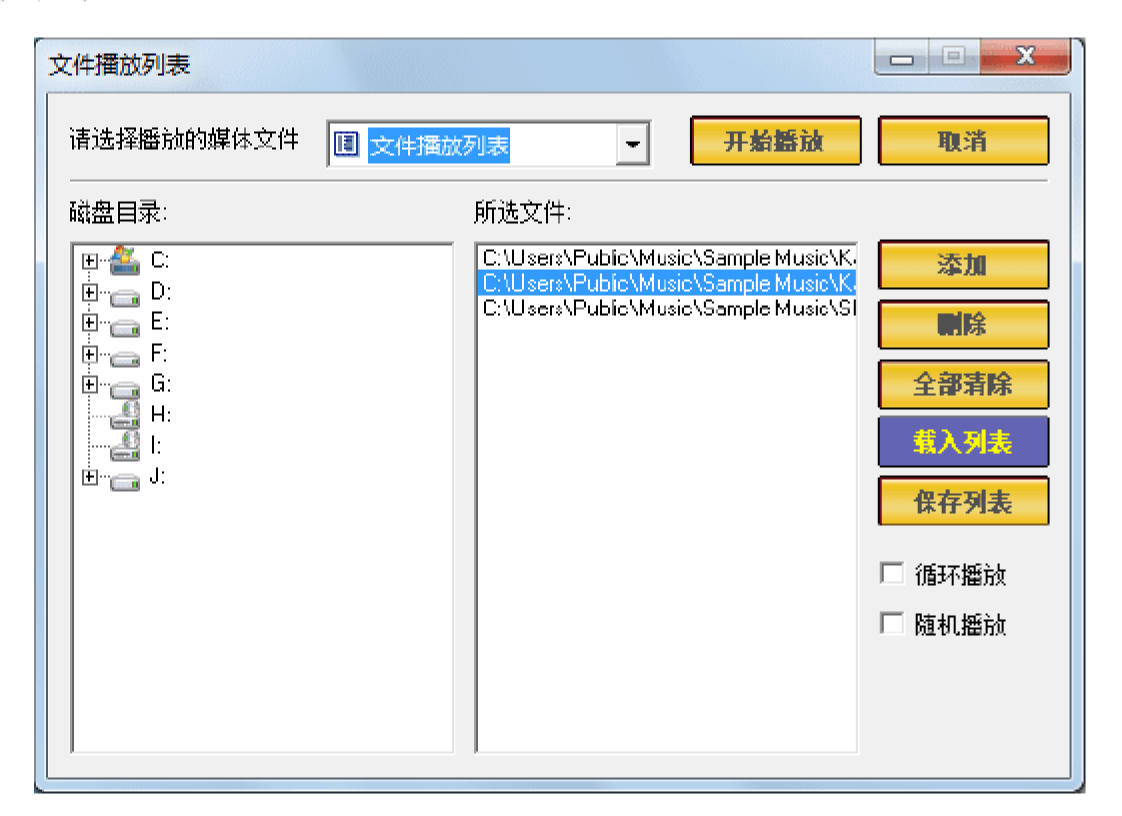


图 8.4　文件播放列表窗口

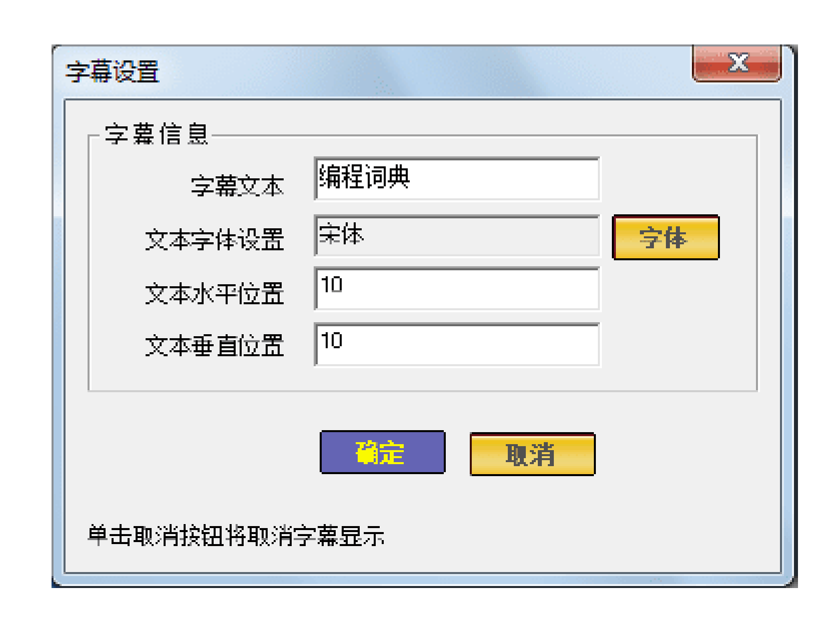


图 8.5　字幕叠加窗口

视频设置窗口如图 8.6 所示。在该模块中，用户可以完成对视频进行色调、饱和度、亮度和对比度的设置。

8.3.4　业务流程图

媒体播放器的业务流程图如图 8.7 所示。



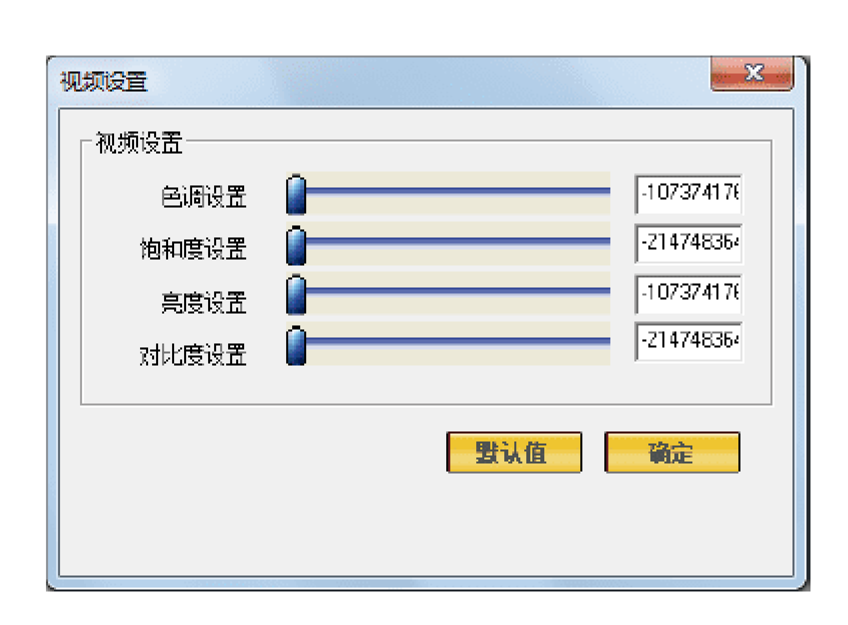


图 8.6　视频设置窗口

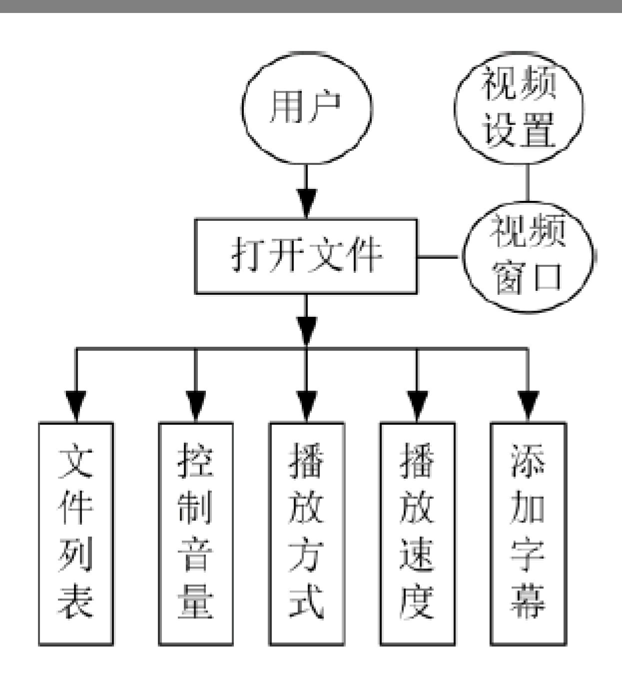


图 8.7　飓风影音业务流程图



8.4　关键技术分析

8.4.1　如何使用 Direct Show 开发包

DirectX 是微软公司推出的一套基于 Windows 平台的图像、声音、输入/输出和网络游戏的编程接口。DirectX 被定义为与设备无关性，即 DirectX 可以使用与设备无关的方法提供设备相关的高性能。DirectX 是一个大家族，其成员主要包括 Direct Input、Direct Play、Direct Setup、Direct Music、Direct Sound、DirectX Media Objects、DirectX Graphics 和 Direct Show 等。其中，Direct Show 主要为 Windows 平台处理媒体文件播放、语音视频采集提供了完整的解决方案。

在使用 Direct Show 之前，需要安装 DirectX 开发包和 Direct Show 开发包。用户可以到微软的官方网站上进行下载。这里需要说明的是，在 DirectX8.1 SDK 版本中包含有 Direct Show SDK，在 DirectX9.0C SDK 的第一个版本 DirectX SDK Summer 2004 中也包含有 Direct Show SDK，之后，在 DirectX SDK 中没有包含 Direct Show SDK，Direct Show SDK 以 Extras 的形式单独发布。

本章程序采用的 DirectX SDK 为 Microsoft DirectX SDK (April 2006)，Direct Show SDK 为 DirectX SDK Extras Frebruary 2005。

在安装完 DirectX SDK 和 Direct Show SDK 之后，为了能够在程序中使用 Direct Show，首先需要将目录下的 BaseTsd.h 文件复制到 Direct Show SDK 安装目录的 Include 目录下。

完成上面的配置之后，还需要引用 dshow.h 头文件，链接 Strmiids 和 quartz 库文件就可以在程序中使用 Direct Show SDK。

```
#include "dshow.h"
#pragma comment (lib,"Strmiids")
#pragma comment (lib,"quartz")
```

8.4.2　使用 Direct Show 开发程序的方法

Direct Show 提供了一个很有用的工具——GraphEit，在 Direct Show 安装目录的 Utilities 目录下。在使用 Direct Show 开发应用程序前，可以使用 GraphEit 预先设置出过滤图表。以播放一个 DAT 文件





为例，使用 graphedt 打开 DAT 文件，GraphEit 将根据系统信息生成过滤图像，如图 8.8 所示。

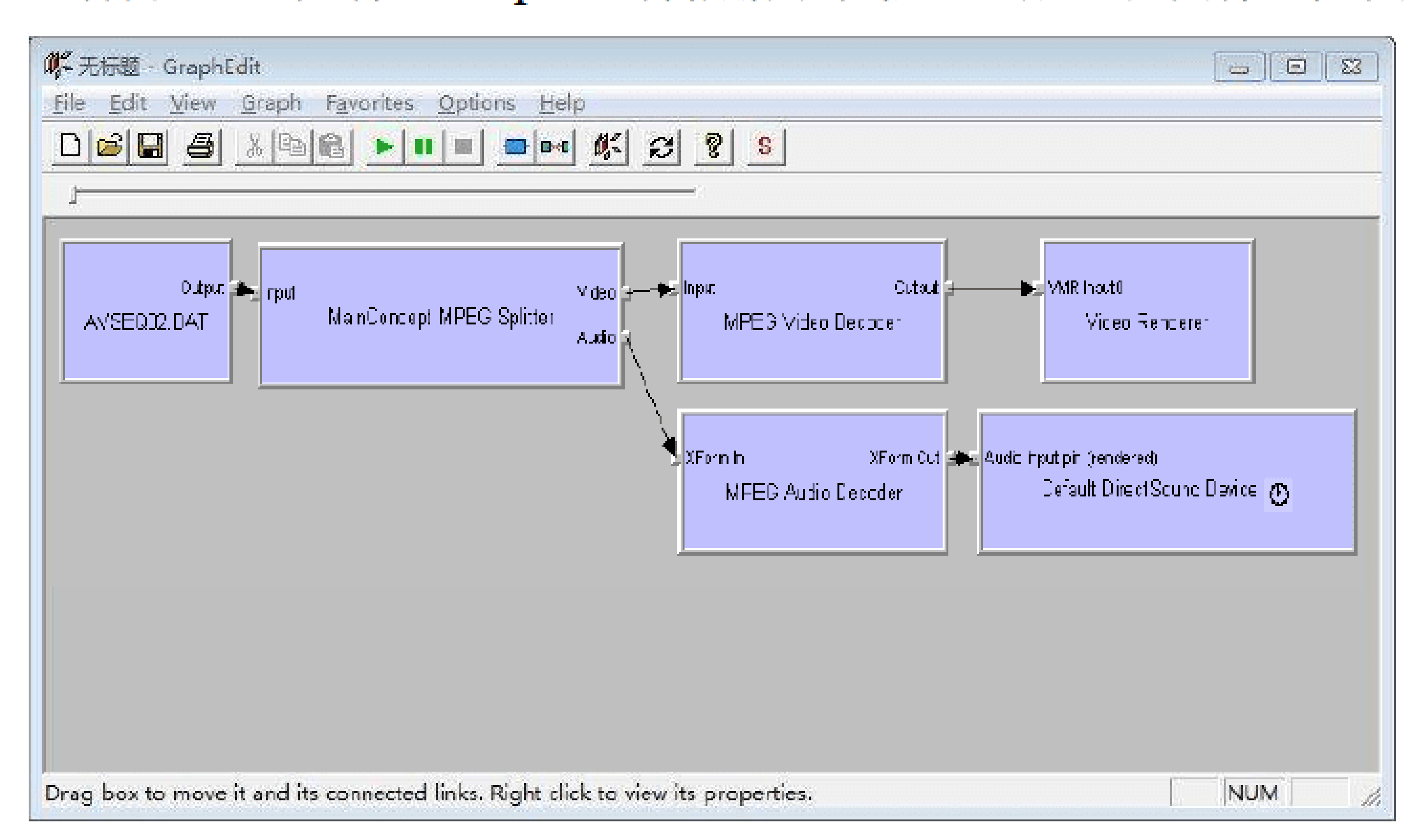


图 8.8　过滤图表

如果需要使用 Direct Show 实现播放 DAT 文件的功能，只要按照图表中显示的过滤器一一创建，并按顺序连接过滤器即可。当然，另一种更简单的方法就是使用过滤图表对象的 RenderFile 方法，该方法将根据文件名自动生成过滤图表。下面给出播放媒体文件的关键代码。

```
IGraphBuilder        *pGraph;                                         //定义　滤图表接口
ICaptureGraphBuilder2 * pBuilder;                                     //定义图表构建接口对象
IMediaControl        *pMediaControl;                                  //定义媒体控制接口
CoCreateInstance(CLSID_CaptureGraphBuilder2,0,CLSCTX_INPROC_SERVER,
     IID_ICaptureGraphBuilder2,(void**)&pBuilder);                    //创建图表构建的接口对象
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
     IID_IFilterGraph2, (void **)&pGraph);                            //创建　滤图表接口对象
pBuilder->SetFiltergraph(pGraph);                                     //设置　滤图表
pGraph->RenderFile(strFile.AllocSysString(),NULL);                    //　媒体文件构建　认的　滤图表
pGraph->QueryInterface(IID_IMediaControl,(void**)&pMediaControl);     //获取媒体控制对象
pMediaControl->Run();                                                 //开始播放文件
```

8.4.3　使用 Direct Show 如何确定媒体文件播放完成

在设计媒体播放器时，需要在媒体文件播放完成后得到通知。因为在设计文件播放列表时，需要在一个媒体文件播放完成后播放下一个文件。在 Direct Show 中，可以通过媒体控制接口和一个单独的线程来实现。具体步骤如下：

（1）创建一个媒体控制接口对象。

```
IMediaEventEx        *pEvent;                                         //定义媒体事件接口对象
pGraph->QueryInterface(IID_IMediaEventEx, (void **)&pEvent);          //获取媒体事件接口对象
```

（2）创建一个线程，执行线程函数，在线程函数中判断媒体文件是否播放完成，如果没有播放完成，发送 CM_POSCHANGE 自定义消息，如果播放完成，发送 CM_COMPLETE 自定义消息。相应代



码如下：

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
    CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter;    //获取主窗口指
    HANDLE   hEvent;                                                  //定义事件句柄
    pWnd->pEvent->GetEventHandle((OAEVENT*) &hEvent);                 //获取事件句柄
    long code,p1,p2;                                                  //定义事件代码参数
    BOOL   done = FALSE;
    while (!done)                                                     //开始执行循环
    {
        pWnd->SendMessage(CM_POSCHANGE);                              //发   CM_POSCHANGE 消息
        if (WaitForSingleObject(hEvent,80)==WAIT_OBJECT_0)            //Direct Show 是否有事件产生
        {
            //获取事件代码
            while (SUCCEEDED(pWnd->pEvent->GetEvent(&code,&p1,&p2,0)))
            {
                pWnd->pEvent->FreeEventParams(code,p1,p2);            //   放事件参数
                if (code==EC_COMPLETE)                                //是否为播放完成
                {
                    pWnd->m_Completed = TRUE;                         //设置播放完成状态
                    pWnd->SendMessage(CM_COMPLETE);                   //发   CM_COMPLETE 消息
                    done=true;                                        //   出循环，结束线程
                }
            }
        }
    }
    return 0;
}
```

8.4.4 使用 Direct Show 行 和播放 度的控制

对于媒体播放软件来说，实现音量调节和播放进度的控制是不可获取的功能。在 Direct Show 中可以使用 IBasicAudio 接口来实现音量的控制。该接口提供了 get_Volume 和 put_Volume 方法用于获取和设置音量。如果想设置为静音，可以将音量设置为-10000L。下面的代码演示了音量的控制。

```
IBasicAudio* pAudio = NULL;                                           //定义 IBasicAudio 接口指
if (pGraph != NULL)
{
    pGraph->QueryInterface(IID_IBasicAudio,(void**)&pAudio);          //获取 IBasicAudio 接口对象
    if (pAudio != NULL)                                               //如果有      数据
    {
        pAudio->get_Volume(&m_lVolumn);                               //获取当前的
        if (m_lVolumn <0)                                             //当前不是最大
        {
            m_lVolumn += 200;                                         //增加
            pAudio->put_Volume(m_lVolumn);                            //设置
        }
```

```
            else
            {
                m_lVolumn = 0;                                        //当前      为最大
            }
        }
}
```

如果想要控制播放的进度，可以使用 IMediaPosition 接口来实现。该接口提供有 get_StopTime 方法获取文件播放结束时的停止事件，提供有 get_CurrentPosition 方法获取当前的播放时间，提供有 put_CurrentPosition 方法设置播放时间，即设置播放进度。有了这些方法，实现播放进度的控制就易如反掌。相应代码如下：

```
IMediaPosition* pPosition = NULL;                                      //定义 IMediaPosition 接口指
if (pGraph != NULL)
{
    pGraph->QueryInterface(IID_IMediaPosition,(void**)&pPosition);     //获取 IMediaPosition 接口对象
    if (pPosition != NULL)
    {
        REFTIME curTime,endTime;
        pPosition->get_StopTime(&endTime);                             //获取播放的停止时
        pPosition->get_CurrentPosition(&curTime);                      //获取当前的播放时
        curTime += 5;                                                  //设置快
        if (curTime <=endTime)
        {
            pPosition->put_CurrentPosition(curTime);                   //设置当前播放的时
        }
        else
        {
            pPosition->put_CurrentPosition(endTime);                   //设置当前播放时   为停止时
        }
    }
}
```

8.4.5　使用 Direct Show 实现字幕叠加

在设计媒体播放器时，添加的一个特色功能之一就是实现字幕叠加功能，效果如图 8.9 所示。



图 8.9　字幕叠加效果

在 Direct Show 中只实现字幕叠加功能，可以有两种方式：一种是自己写一个字幕叠加的过滤器，将其插入到视频解码过滤器和视频显示过滤器之间，这样可以实现字幕叠加功能，但是在设计亮度、饱和度和对比度调节功能时发现字幕叠加过滤器与其产生冲突；而采用另一种方式，即使用 VideoMixingRenderer9 过滤器实现视频显示，笔者会将实现字幕叠加功能的过滤器源代码放置在本书资源包中。下面介绍使用 VideoMixingRenderer9 过滤器实现字幕叠加。默认情况下，Direct Show 使用 Video Renderer 过滤器来显示视频图像，该过滤器没有实现字幕叠加功能，需要将该过滤器替换为 VideoMixingRenderer9 过滤器，VideoMixingRenderer9 过滤器支持 IVMRMixerBitmap9 接口，该接口提供了 SetAlphaBitmap 方法可以实现字幕叠加功能。主要代码如下：

```
CoCreateInstance(CLSID_VideoMixingRenderer9, NULL, CLSCTX_ALL,
          IID_IBaseFilter, (void **)&pRender);                    //创建 VideoMixingRenderer9  滤器
```

实现使用 VideoMixingRenderer9 过滤器替换默认的视频显示过滤器的相应代码如下。

```
IBaseFilter *pRenderFiler = NULL;                                  //定义 IBaseFilter 接口对象
//获取 IBaseFilter 接口对象
pWnd->pGraph->FindFilterByName(L"Video Renderer",(IBaseFilter**)&pRenderFiler);
if (pRenderFiler!=NULL)                                            //包含视  信息
{
     pWnd->m_bViewPlay = TRUE;
     IPin* pVideoIn = NULL;
     pVideoIn   = pWnd->FindPin(pRenderFiler,PINDIR_INPUT);        //查找  入引脚
     if (pVideoIn)
     {
          pVideoIn->Disconnect();                                  //先断开  入引脚与视  解码的  接
     }
     pWnd->pGraph->RemoveFilter(pRenderFiler);                     //移   认的视  显示  滤器
     pWnd->pGraph->AddFilter(pWnd->pRender,L"Render");             //添加 VideoMixingRenderer9  滤器
     //获取视  解码器
     pWnd->pGraph->FindFilterByName(L"MPEG Video Decoder",(IBaseFilter**)&pWnd->pBase);
     if (pWnd->pBase)
     {
          IPin* pOutPin = NULL;                                    //定义  出引脚接口指
          //获取视  解码的  出引脚
          pOutPin = pWnd->FindPin(pWnd->pBase,PINDIR_OUTPUT);
          if (pOutPin != NULL)
          {
               IPin* pColorIn = NULL;
               IPin* pColorOut = NULL;
               //获取  入和  出引脚
               IPin *pTextIn = NULL;
               IPin *pTextOut = NULL;
               IPin *pRenderIn = NULL;
               //查找  入引脚
               pRenderIn = pWnd->FindPin(pWnd->pRender,PINDIR_INPUT);
               HRESULT hRet = 0;
               hRet = pOutPin->Disconnect();                       //断开  出引脚
```

```
                // 接视  解码的  出引脚与 VideoMixingRenderer9  滤器的  入引脚
                hRet = pWnd->pGraph->ConnectDirect(pOutPin,pRenderIn,NULL);
            }
        }
}
else
{
    pWnd->m_bViewPlay = FALSE;                              //没有视  信息
}
```

利用 VideoMixingRenderer9 过滤器实现字幕叠加功能。相应代码如下：

```
IVMRMixerBitmap9 * pBmp9 = NULL;                            //定义 IVMRMixerBitmap9 接口指
pRender->QueryInterface(IID_IVMRMixerBitmap9,(void**)&pBmp9); //获取 IVMRMixerBitmap9 接口对象
if (pBmp9!= NULL)
{
    COverlayText OverlayDlg;                                //定义字幕叠加对话框
    if (OverlayDlg.DoModal()==IDOK)                         //显示字幕叠加对话框
    {
        BYTE byR,byG,byB;                                   //定义  色变
        byR = GetRValue(OverlayDlg.m_TextColor);            //获取文本  色
        byG = GetGValue(OverlayDlg.m_TextColor);
        byB = GetBValue(OverlayDlg.m_TextColor);
        LOGFONT logfont = OverlayDlg.m_LogFont;             //获取字体信息
        int nX = OverlayDlg.m_HorPos;                       //获取坐标
        int nY = OverlayDlg.m_VerPos;
        IVideoWindow * pVideoWnd = NULL;                    //定义 IVideoWindow 接口指
        long lVideoWidth, lVideoHeight;
        //获取 IVideoWindow 接口指
        pRender->QueryInterface(IID_IVideoWindow, (void**)&pVideoWnd);
        if (pVideoWnd!= NULL)
        {
            pVideoWnd->get_Width(&lVideoWidth);             //获取视  窗口的宽度
            pVideoWnd->get_Height(&lVideoHeight);           //获取视  窗口的  度
            //创建一个设备上下文
            HDC hBmpDC = CreateCompatibleDC(GetDC()->m_hDC);
            CFont Font;                                     //定义字体对象
            Font.CreateFontIndirect(&logfont);              //创建字体
            //  中字体对象
            HFONT hOldFont = (HFONT) SelectObject(hBmpDC,Font.m_hObject);
            int nLength, nTextBmpWidth, nTextBmpHeight;
            SIZE szText={0};
            nLength = strlen(OverlayDlg.m_Text);            //获取字符串的  度
            //获取文本的  度和  度
            GetTextExtentPoint32(hBmpDC, OverlayDlg.m_Text, nLength, &szText);
            nTextBmpHeight = szText.cy;
            nTextBmpWidth    = szText.cx;
            HBITMAP hBmp = CreateCompatibleBitmap(GetDC()->m_hDC,
                nTextBmpWidth, nTextBmpHeight);             //创建位图
```

```
            BITMAP bmObj;                                          //定义位图信息对象
            HBITMAP hbmOld;                                        //定义位图句柄
            GetObject(hBmp, sizeof(bmObj), &bmObj);                //获取位图信息
            hbmOld = (HBITMAP)SelectObject(hBmpDC, hBmp);          //   中位图
            //定义文本的矩形区域
            RECT rcText;
            SetRect(&rcText, 0, 0, nTextBmpWidth, nTextBmpHeight);
            //设置背景   色，注意与文本   色接   ，否则效果不好
            SetBkColor(hBmpDC, RGB(byR, byG, byB-1));
            SetTextColor(hBmpDC, RGB(byR, byG, byB));              //设置文本   色
            TextOut(hBmpDC, 0, 0,OverlayDlg.m_Text, nLength);      //   出文本
            VMR9AlphaBitmap bmpInfo;                               //定义参数
            ZeroMemory(&bmpInfo, sizeof(bmpInfo));                 //初始化参数
            bmpInfo.dwFlags = VMRBITMAP_HDC;                       //设置参数标记
            bmpInfo.hdc = hBmpDC;                                  //设置设备上下文
            //等比例   出文字
            double xRate = (double)nTextBmpWidth /lVideoWidth;
            double yRate = (double)nTextBmpHeight / lVideoHeight;
            double fX = (double)nX / lVideoWidth;                  //确定文本   出比例
            double fY = (double)nY / lVideoHeight;
            bmpInfo.rDest.left   = fX;
            bmpInfo.rDest.right = fX+xRate;
            bmpInfo.rDest.top = fY;
            bmpInfo.rDest.bottom = fY+ yRate;
            bmpInfo.rSrc = rcText;                                 //设置文本显示区域
            bmpInfo.clrSrcKey = RGB(byR, byG, byB-1);              //设置关      色
            bmpInfo.dwFlags |= VMRBITMAP_SRCCOLORKEY;
            bmpInfo.fAlpha = 1.0;
            pBmp9->SetAlphaBitmap(&bmpInfo);                       //实现图像字幕叠加
        }
    }
}
```

8.4.6 使用 Direct Show 实现亮度、　和度和对比度调节

在媒体播放器中，实现了对视频图像的亮度、饱和度和对比度的设置。主要方法是使用 VideoMixingRenderer9 过滤器支持的 IVMRMixerControl9 接口实现的，该接口提供了 SetProcAmpControl 方法用于设置图像的亮度、饱和度和对比度等信息。主要代码如下：

```
void CDirectShowEventDlg::SetViewInfo(int nFlag, float fValue)
{
    IVMRMixerControl9 * pControl = NULL;                       //定义 IVMRMixerControl9 接口指
    if (pRender != NULL)
    {
        //获取 IVMRMixerControl9 接口对象
        pRender->QueryInterface(IID_IVMRMixerControl9,(void**)&pControl);
        if (pControl != NULL)
```

```
        {
            VMR9ProcAmpControl vmrParam;                              //定义参数
            memset(&vmrParam,0,sizeof(VMR9ProcAmpControl));           //初始化参数
            vmrParam.dwSize = sizeof(VMR9ProcAmpControl);             //设置参数大小
            vmrParam.dwFlags = nFlag;                                 //设置标记
            vmrParam.Brightness = fValue;                             //设置亮度值
            vmrParam.Contrast = fValue;                               //设置对比度值
            vmrParam.Hue = fValue;                                    //设置色调值
            vmrParam.Saturation = fValue;                             //设置　和度值
            pControl->SetProcAmpControl(0,&vmrParam);                 //设置视　图像　色信息
        }
    }
}
```

8.4.7　设计显示目录和文件的树视图控件

在设计本模块的文件播放列表功能时，需要以树结构显示磁盘目录和文件信息。但是，在 Visual C++中没有现成的控件可以使用，为此，设计了一个显示磁盘目录和文件的树控件，效果如图 8.10 所示。

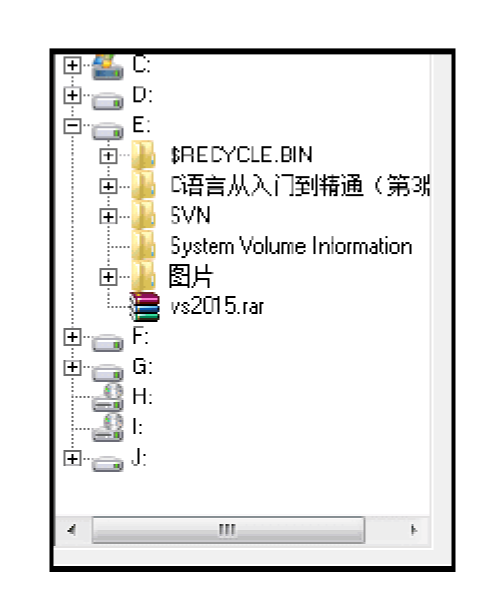


图 8.10　显示目录和文件的树控件

树控件的主要设计思路是首先获取系统的磁盘目录，将其作为根节点添加到树控件中，然后向每个根节点下添加一个空的子节点，作用是让根节点显示一个“+”符号，表示有子目录或文件。当用户单击某一个根节点要展开子节点时，首先删除空的子节点，然后遍历该目录下的子目录或文件（注意是直接子目录和文件，而不是该目录下的所有目录和文件），而用户单击“-”符号收缩子节点时将删除该目录的所有节点，然后添加一个空的子节点。至于各个节点的图标，可以使用 SHGetFileInfo 函数来获取系统关联的文件图标。树控件的具体设计步骤如下：

（1）从 CTreeCtrl 类派生一个子类——CSysTreeCtrl，向该类中添加成员变量。

```
CImageList m_ImageList;                                       //图像列表
```

（2）向 CSysTreeCtrl 类中添加 IsSubDir 方法，作用是判断当前的目录是否有子目录或文件。如果有子目录文件，在添加该节点时将为其插入一个空的子节点以显示“+”符号，表示有子目录或文件。实现代码如下：

```
BOOL CSysTreeCtrl::IsSubDir(LPCTSTR strPath)
{
    CFileFind flFind;                                         //定义文件查找对象
    CString csPath = strPath;
    BOOL bFind = FALSE;
    //保证　径以\*.*结尾
    if (_tcslen(strPath) == 3)
    {
```

```
        if (strPath[1] == ':' && strPath[2] =='\\')                  //判断字符串结尾字符
            csPath += "*.*";
        else
            csPath += "\\*.*";
    }
    else
    {
        csPath += "\\*.*";
    }
    bFind = flFind.FindFile(csPath);                                 //开始查找文件
    while (bFind)
    {
        bFind = flFind.FindNextFile();                               //查找下一个文件
        if (!flFind.IsDots())                                        //判断是否有子目录或文件
        {
            return TRUE;
        }
    }
    return FALSE;
}
```

（3）向 CSysTreeCtrl 类中添加 EnumSysDir 方法，列举系统磁盘目录。实现代码如下：

```
void CSysTreeCtrl::EnumSysDir()
{
    char    pchDrives[128] = {0};                                    //定义字符数组
    char* pchDrive;                                                  //定义字符指
    GetLogicalDriveStrings(sizeof(pchDrives), pchDrives);           //获取磁盘目录
    pchDrive = pchDrives;
    while(*pchDrive)
    {
        HTREEITEM hParent = AddItem(TVI_ROOT, pchDrive);            //向树控件中添加根节点
        if (IsSubDir(pchDrive))                                      //是否有子目录或文件
            InsertItem("", 0, 0, hParent);                           //插入空的节点
        pchDrive += strlen(pchDrive) + 1;                            //获取下一个磁盘目录
    }
}
```

（4）向 CSysTreeCtrl 类中添加 ExtractPath 方法，当遍历目录或文件时，将获取目录或文件的完整名称，包含详细的路径信息，该方法的作用就是只显示相对的目录和文件名，不包含路径。实现代码如下：

```
CString CSysTreeCtrl::ExtractPath(LPCTSTR strPath)
{
    CString csPath = "";
    int nPos;
    csPath = strPath;
    if (csPath.Right(1) == '\\')                                     //   去结尾的"\"
    {
```

```
        csPath.SetAt(csPath.GetLength() - 1, '\0');
    }
    nPos = csPath.ReverseFind('\\');                        //反向查找“\”字符
    if (nPos != -1)
        csPath = csPath.Mid(nPos + 1,csPath.GetLength());   //截取字符串
    return (LPCTSTR)csPath;                                 //  回目录或文件名
}
```

（5）向 CSysTreeCrl 类中添加 GetFullPath 方法，该方法的作用是返回某一节点对应的完整的目录或文件名。实现代码如下：

```
CString CSysTreeCtrl::GetFullPath(HTREEITEM hItem)
{
    CString csRet;                                  //记录目录
    CString csCurDir;                               //当前目录
    HTREEITEM hParent = hItem;
    csRet = "";
    while (hParent)                                 //  历父节点
    {
        csCurDir  = GetItemText(hParent);           //获取当前节点文本
        csCurDir += "\\";                           //结尾添加“\”符号
        csRet = csCurDir + csRet;                   //累计目录
        hParent = GetParentItem(hParent);           //获取上级节点
    }
   if (csRet.Right(1) == '\\')                      //  去结尾的“\”符号
        csRet.SetAt(csRet.GetLength() - 1, '\0');
   return csRet;                                    //  回结果
}
```

（6）向 CSysTreeCtrl 类中添加 GetFileImage 方法，获取系统的文件图像列表。实现代码如下：

```
BOOL CSysTreeCtrl::GetFileImage()
{
    SHFILEINFO shInfo;                              //定义外壳文件信息对象
    memset(&shInfo,0,sizeof(SHFILEINFO));           //初始化外壳文件信息对象
    HIMAGELIST hImage = NULL;                       //定义图像列表句柄
    //获取系统文件图像列表
    hImage = (HIMAGELIST)SHGetFileInfo("C:\\",0,&shInfo, sizeof(SHFILEINFO),
        SHGFI_SYSICONINDEX | SHGFI_SMALLICON);
    if (!hImage)
    {
        return FALSE;
    }
    m_ImageList.Attach(hImage);                     //  加系统文件图像列表
    SetImageList(&m_ImageList, TVSIL_NORMAL);       //设置树控件的图像列表
    return TRUE;
}
```

（7）向 CSysTreeCtrl 中添加 LoadPath 方法，将根据某一个目录加载其直接子目录和文件，不包



括间接子目录和文件。实现代码如下：

```
void CSysTreeCtrl::LoadPath(HTREEITEM hParent, LPCTSTR strPath)
{
	CFileFind flFind;                                   //定义文件查找对象
	CString csPath = strPath;
	BOOL bFind;
	CSortList strDirArray;                              //定义字符串数组，用于排序和记录目录
	CSortList strFileArray;                             //定义字符串数组，用于排序和记录文件名
	if (csPath.Right(1) != "\\")                        //在目录结尾添加\*.*符号
		csPath += "\\";
	csPath += "*.*";
	bFind = flFind.FindFile(csPath);                    //开始查找文件
	while (bFind)
	{
		bFind = flFind.FindNextFile();                  //查找下一个文件
		if (flFind.IsDirectory() && !flFind.IsDots())   //判断是否为目录
		{
			strDirArray.Add(flFind.GetFilePath());      //记录目录名
		}
		if (!flFind.IsDirectory())                      //是否为文件
			strFileArray.Add(flFind.GetFilePath());     //记录文件名
	}
	strDirArray.Sort();                                 //对目录    行排序
	SetRedraw(FALSE);
	CWaitCursor wait;
	for (int i = 0; i < strDirArray.GetSize(); i++)     //将目录添加到树控件中
	{
			HTREEITEM hItem = AddItem(hParent, strDirArray.GetAt(i));
			if (IsSubDir(strDirArray.GetAt(i)))
				InsertItem("", 0, 0, hItem);            //添加空节点
	}
	strFileArray.Sort();                                //对文件名    行排序
	for (i = 0; i < strFileArray.GetSize(); i++)        //将文件名添加到树控件中
	{
		HTREEITEM hItem = AddItem(hParent, strFileArray.GetAt(i));
	}
	SetRedraw(TRUE);
}
```

（8）向 CSysTreeCtrl 中添加 LoadTree 方法，加载磁盘根目录到树节点中。实现代码如下：

```
BOOL CSysTreeCtrl::LoadTree(LPCTSTR strPath)
{
	DWORD dwStyle = GetStyle();                         //获取控件    格
	if (dwStyle & TVS_EDITLABELS)
	{
		ModifyStyle(TVS_EDITLABELS, 0);                 //设置控件    格
	}
	DeleteAllItems();                                   //删    所有的树节点
```



```
        if (!GetFileImage())                                //获取系统文件图像列表
            return FALSE;
        if (strPath == NULL || strPath[0] == '\0')
        {
            EnumSysDir();                                   //加　系统磁盘目录
        }
        return TRUE;
}
```

（9）向 CSysTreeCtrl 类中添加 ExpandItem 方法，该方法的作用是展开某一个子节点，显示子节点。实现代码如下：

```
void CSysTreeCtrl::ExpandItem(HTREEITEM hItem, UINT nCode)
{
        CString strPath;
        if (nCode == TVE_EXPAND)                            //如果为展开节点
        {
            HTREEITEM hChild = GetChildItem(hItem);         //获取子节点
            while (hChild)
            {
                DeleteItem(hChild);                         //第一个　目为空，目的是显示加号
                hChild = GetChildItem(hItem);               //获取下一个子节点
            }
            strPath = GetFullPath(hItem);                   //获取完整目录
            LoadPath(hItem, strPath);                       //加　所有子目录和文件
        }
}
```

（10）处理树控件的 TVN_ITEMEXPANDED 反射消息，当用户展开或收缩节点时显示或者删除节点。实现代码如下：

```
void CSysTreeCtrl::OnItemExpanded(NMHDR *pNMHDR, LRESULT *pResult)
{
        NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
        CString strPath;
        if (pNMTreeView->itemNew.state & TVIS_EXPANDED)     //如果为展开节点
        {
            ExpandItem(pNMTreeView->itemNew.hItem, TVE_EXPAND);  //展开所有子节点
        }
        else                                                //收缩节点
        {
            HTREEITEM hChild = GetChildItem(pNMTreeView->itemNew.hItem);     //获取子节点
            while (hChild)                                  //利用循环删　所有子节点
            {
                DeleteItem(hChild);                         //删　节点
                hChild = GetChildItem(pNMTreeView->itemNew.hItem);  //获取下一个子节点
            }
            InsertItem("", pNMTreeView->itemNew.hItem);     //插入空的子节点
        }
```

```
        *pResult = 0;
}
```

至此，CSysTreeCtrl 控件设计完成。



8.5 媒体播放器主窗口设计

8.5.1 媒体播放器主窗口概

媒体播放器主窗口主要用于显示媒体文件的播放进度、播放时间等，并控制播放进度、音量大小以及调用字幕叠加、视频设置等子窗口。媒体播放器主窗口效果如图 8.11 所示。

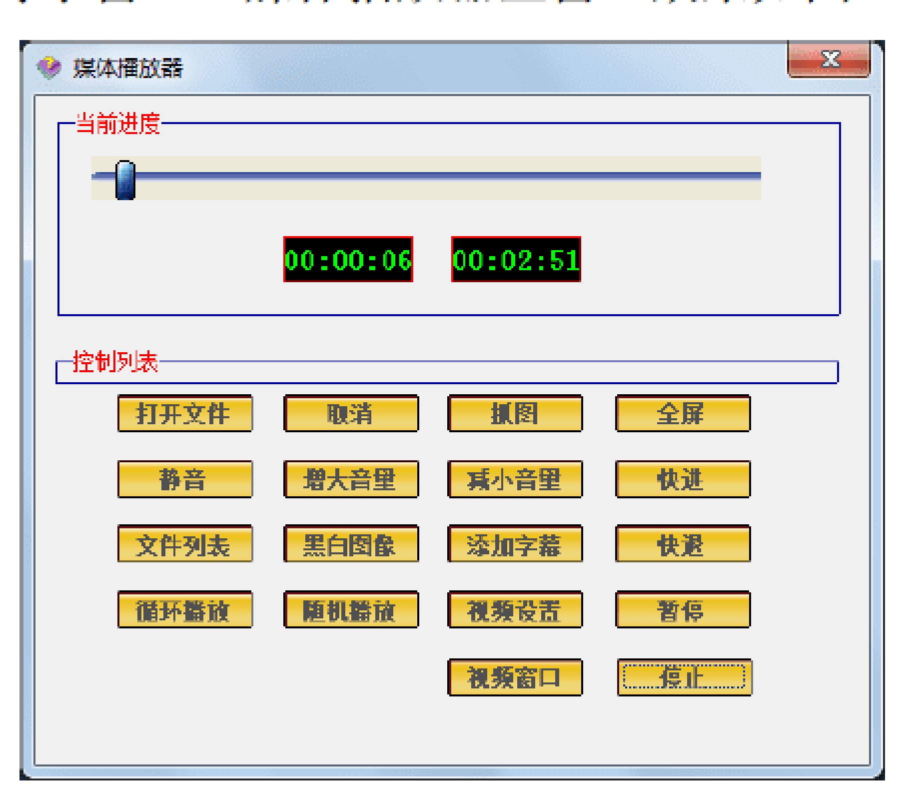


图 8.11 媒体播放器主窗口

8.5.2 媒体播放器主窗口界 设计

媒体播放器主窗口界面设计过程如下：

（1）创建一个对话框类，类名为 CDirectShowEventDlg。

（2）向对话框中添加按钮、静态文本和滑块控件。

（3）设置主要控件属性，如表 8.1 所示。

表 8.1 媒体播放器主窗口控件属性设置

控件 ID	控 件 属 性	关 联 变
IDC_CTLLIST	Caption：控制列表 Border：FALSE	CCustomGroup：m_CtlList
IDC_GRAY	Caption：黑白图像	CButton：m_GrayBtn
IDC_PROCESSCTRL	默认	CCustomSlider：m_ProgressCtrl
IDC_CURPOS	Caption：空 Border：FALSE	CNumLabel：m_CurPos



8.5.3　媒体播放器主窗口实现　程

媒体播放器主窗口实现过程如下：

（1）引用 Direct Show 相关头文件和库文件。代码如下：

```
#include "dshow.h"
#include "D3d9.h"
#include "vmr9.h"
#include "Objbase.h"
#pragma comment (lib,"Strmiids")
#pragma comment (lib,"quartz")
```

（2）在应用程序初始化时初始化 Com 库，因为 Direct Show 操作是基于 Com 技术实现的。

```
CoInitialize(NULL);                                        //初始化 Com 库
```

（3）向对话框中添加 FindPin 方法，用于查找某一过滤器的输入、输出引脚。相应代码如下：

```
//查找引脚
IPin* CDirectShowEventDlg::FindPin(IBaseFilter *pFilter, PIN_DIRECTION dir)
{
    IEnumPins* pEnumPins;                                  //定义 IEnumPins 接口指
    IPin* pOutpin;                                         //定义 IPin 接口指
    PIN_DIRECTION pDir;                                    //定义引脚方向
    pFilter->EnumPins(&pEnumPins);                         //列举  滤器引脚
    while (pEnumPins->Next(1,&pOutpin,NULL)==S_OK)
    {
        pOutpin->QueryDirection(&pDir);                    //获取引脚方向（  入或  出引脚）
        if (pDir==dir)                                     //判断引脚方向
        {
            return pOutpin;
        }
    }
}
```

（4）向对话框中添加 Done 方法，当用户播放一个媒体文件时，将创建一个线程，用于检测媒体文件是否播放完成，如果播放完成，将向主窗口发送自定义消息 CM_COMPLETE，该消息关联 Done 方法。Done 方法的作用是在媒体文件播放完成之后终止线程，隐藏 Direct Show 的视频显示窗口，释放媒体控制接口对象指针，释放过滤图标，恢复对话框的成员变量为初始状态，修改视频显示窗口，使得在媒体文件播放完成后不可以调整窗口大小。实现代码如下：

```
void CDirectShowEventDlg::Done(WPARAM wParam, LPARAM lParam)
{
    if (m_hThread)                                         //判断线程是否  行
    {
        TerminateThread(m_hThread,0);                      //终止线程
```

```
        m_hThread = NULL;
    }
    if (pMediaControl != NULL)                            //是否正在播放
    {
        pMediaControl->Stop();                            //停止播放
        m_bStop = TRUE;
        if (pViewWnd != NULL)                             //是否显示视  图像
        {
            pViewWnd->put_Visible(FALSE);                 //  藏视  窗口
        }
    }
    if (pMediaControl)                                    //判断媒体控制接口指  是否为空
    {
        pMediaControl->Release();                         //  放媒体控制接口指
        pMediaControl = NULL;
    }
    if (pGraph)                                           //判断  滤图表接口指  是否为空
    {
        pGraph->Release();                                //  放  滤图表接口指
        pGraph = NULL;
    }
    if (pEvent)
    {
        pEvent->Release();                                //  放媒体控制对象
        pEvent = NULL;
    }
    m_bFullScreen = FALSE;                                //将成员变  恢复为原来的状态
    pViewWnd = NULL;
    pBaseVideo = NULL;
    pBase = NULL;
    m_bViewPlay = FALSE;
    m_lVolumn = 0;
    m_bMute = FALSE;
    m_bSpeed = FALSE;
    m_bBack= FALSE;
    m_bGrayImage = FALSE;
    m_fSaturation = m_fBright = m_fContrast = m_fHue = 1;
    pRender = NULL;
    m_bStop = m_bPause = FALSE;
    m_hThread = NULL;
    m_Stop.SetWindowText("停止");                         //设置“停止”按  的文本
    m_GrayBtn.SetWindowText("  白图像");
    m_Pause.SetWindowText("暂停");
    m_Progress.SetText("00:00:00");
    m_CurPos.SetText("00:00:00");
    m_ProgressCtrl.SetPos(0);
    //播放完成
    m_Previewed = FALSE;
    m_Completed = TRUE;
```

```
        m_DisplayWnd.ModifyStyle(WS_SIZEBOX,0);                    //修改视　显示窗口的　格
        m_DisplayWnd.m_Panel.ModifyStyle(SS_BLACKRECT,SS_BITMAP);
        m_DisplayWnd.m_Panel.SetBitmap(m_DisplayWnd.bmp);          //设置　认的位图
        m_DisplayWnd.SetWindowPos(NULL,0,0,m_OriginRC.Width(),
            m_OriginRC.Height(),SWP_NOMOVE);                       //恢复视　窗口大小
}
```

（5）向对话框中添加 OnPosChange 方法，用于显示当前的播放时间，当播放一个媒体文件时，将创建一个线程判断媒体文件是否播放完成，如果没有播放完成，则发送自定义消息 CM_POSCHANGE，该消息关联 OnPosChange 方法。相应代码如下：

```
void CDirectShowEventDlg::OnPosChange()
{
    if (pGraph != NULL)
    {
        IMediaPosition* pPosition = NULL;                          //定义 IMediaPosition 接口指
        //获取 IMediaPosition 接口对象
        pGraph->QueryInterface(IID_IMediaPosition,(void**)&pPosition);
        if (pPosition != NULL)
        {
            REFTIME endTime;
            pPosition->get_CurrentPosition(&endTime);              //获取当前的播放时
            m_ProgressCtrl.SetPos(endTime);                        //设置　度条的显示位置
            int nHour = endTime / 3600;                            //将秒　换为小时:分:秒的形式
            int nMinute = (endTime - nHour*3600)/60;
            int nSecond = (int)endTime % 60;
            CString csTime,csSpace;
            csSpace = "";
            if (nHour<10)
                csSpace += "0%d:";
            else
                csSpace += "%d:";
            if (nMinute<10)
                csSpace += "0%d:";
            else
                csSpace += "%d:";
            if (nSecond<10)
                csSpace += "0%d";
            else
                csSpace += "%d";
            csTime.Format(csSpace,nHour,nMinute,nSecond);         //格式化字符串
            m_CurPos.SetText(csTime);                              //显示播放时
        }
    }
}
```

（6）向对话框中添加 PlayFile 方法，播放指定的媒体文件。该方法首先判断当前是否正在播放媒体文件，如果是则调用 Done 方法结束媒体文件的播放，然后构建适当的过滤图表，在构建完过滤图表

之后，将创建一个线程，该线程的作用是修改过滤图表，默认情况下，Direct Show 使用 Video Renderer 过滤器来显示视频图像，该过滤器不能实现字幕叠加功能，不能够调节视频图像的亮度、饱和度和对比度，需要将该过滤器替换为 VideoMixingRenderer9 过滤器，线程函数的作用就是使用 VideoMixingRenderer9 过滤器替换 Video Renderer 过滤器。接着在 PlayFile 方法中将判断媒体文件是否包含视频显示，如果是则显示视频窗口，最后获取媒体文件的长度，创建一个单独的线程来检测媒体文件是否播放完成，实现代码如下：

```
void CDirectShowEventDlg::PlayFile(LPCTSTR lpFileName)
{
	if (pGraph != NULL)                                         //之前已经播放文件或者正在播放文件
	{
		Done(0,0);                                              //停止播放
	}
	m_bStop = FALSE;                                            //初始化变
	pBuilder = NULL;
	pGraph = NULL;
	pMediaControl = NULL;
	m_FileName = lpFileName;
	CoCreateInstance(CLSID_CaptureGraphBuilder2,0,CLSCTX_INPROC_SERVER,
		IID_ICaptureGraphBuilder2,(void**)&pBuilder);           //创建图表构建接口对象
	CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
		IID_IFilterGraph2, (void **)&pGraph);                   //创建   滤图表对象
	pBuilder->SetFiltergraph(pGraph);                           //构     认的   滤图表
	m_bInvalidFile = FALSE;
	pRender = NULL;
	CoCreateInstance(CLSID_VideoMixingRenderer9, NULL, CLSCTX_ALL,
		IID_IBaseFilter, (void **)&pRender);                    //创建 VideoMixingRenderer9    滤器
	m_ThreadStop = FALSE;
	//创建一个单独的线程实现视   显示   滤器的替换
	HANDLE hHandle = CreateThread(NULL,0,PlayVideoFile,(void*)this,0,NULL);
	while (!m_ThreadStop)                                       //等待线程函数执行完成
	{
		MSG msg;
		::GetMessage(&msg,NULL,0,WM_USER);                      //在循环   程中响应界   操作
		::TranslateMessage(&msg);
		::DispatchMessage(&msg);
	}
	if (m_bInvalidFile)                                         //文件   法
	{
		Done(0,0);                                              //停止播放
		return;
	}
	pViewWnd= NULL;
	//获取   览窗口
	pGraph->QueryInterface(IID_IVideoWindow,(void**)&pViewWnd);
	if (pViewWnd)
	{
		//设置   览窗口的拥有者
```

```
        pViewWnd->put_Owner((long)m_DisplayWnd.m_Panel.m_hWnd);
        pViewWnd->put_Left(0);
        pViewWnd->put_Top(0);
        //获取　览窗口　格
        long style;
        pViewWnd->get_WindowStyle(&style);
        style = style & ~WS_CAPTION;
        style = style & ~WS_DLGFRAME;
        style = style & WS_CHILD;
        pViewWnd->put_WindowStyle(style);
        //设置　览窗口的宽度和　度
        CRect rc;
        m_DisplayWnd.m_Panel.GetClientRect(rc);
        pViewWnd->put_Height(rc.Height());
        pViewWnd->put_Width(rc.Width());
        pViewWnd->put_MessageDrain((OAHWND)m_DisplayWnd.m_Panel. m_hWnd );
    }
    if (m_bViewPlay)
    {
        m_DisplayWnd.ShowWindow(SW_SHOW);                //显示视　显示窗口
        m_DisplayWnd.ModifyStyle(0,WS_SIZEBOX);          //设置视　显示窗口　格
    }
    else
    {
        m_DisplayWnd.ShowWindow(SW_HIDE);                //　藏视　显示窗口
        m_DisplayWnd.ModifyStyle(WS_SIZEBOX,0);          //设置视　显示窗口　格
    }
    m_hThread = NULL;
    //获取媒体控制接口对象
    pGraph->QueryInterface(IID_IMediaControl,(void**)&pMediaControl);
    pMediaControl->Run();                                //开始播放文件
    IMediaPosition* pPosition = NULL;                    //定义 IMediaPosition 接口指
    //获取 IMediaPosition 接口对象
    pGraph->QueryInterface(IID_IMediaPosition,(void**)&pPosition);
    if (pPosition != NULL)
    {
        REFTIME curTime,endTime;
        pPosition->get_StopTime(&endTime);               //获取文件播放的停止时
        pPosition->get_CurrentPosition(&curTime);        //获取文件播放的当前时
        m_ProgressCtrl.SetRange(curTime,endTime);        //设置　度条的范围
            //将秒　换为小时:分:秒的形式
        int nHour = endTime / 3600;                      //获取小时
        int nMinute = (endTime - nHour*3600)/60;         //获取分
        int nSecond = (int)endTime % 60;                 //获取秒
        CString csTime,csSpace;
        csSpace = "";
        if (nHour<10)
            csSpace += "0%d:";
        else
```

```
            csSpace += "%d:";
        if (nMinute<10)
            csSpace += "0%d:";
        else
            csSpace += "%d:";
        if (nSecond<10)
            csSpace += "0%d";
        else
            csSpace += "%d";
        csTime.Format(csSpace,nHour,nMinute,nSecond);          //格式化字符串
        m_Progress.SetText(csTime);                            //设置显示时
    }
    pEvent = NULL;
    //获取 IID_IMediaEventEx 接口对象
    pGraph->QueryInterface(IID_IMediaEventEx, (void **)&pEvent);
    m_Completed = FALSE;
    DWORD threadID;
    //开始一个线程，检测文件是否播放完成
    m_hThread =  CreateThread(NULL,0,ThreadProc,(void*)this,0,&threadID);
    m_Previewed = TRUE;
}
```

（7）添加 PlayVideoFile 全局函数，该函数作为一个线程函数，当构建完媒体文件的过滤图表之后，将创建一个线程执行线程函数 PlayVideoFile，用于使用 VideoMixingRenderer9 过滤器替换 Video Renderer 过滤器。相应代码如下：

```
DWORD WINAPI PlayVideoFile(LPVOID lpParameter)
{
    //获取主窗口指
    CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter;
    CString strFile= pWnd->m_FileName;                         //获取播放的文件名
    //构建  滤图标
    HRESULT hRet = pWnd->pGraph->RenderFile(strFile.AllocSysString(),NULL);
    if(hRet!=S_OK)
    {
        pWnd->m_bInvalidFile = TRUE;                           //  法的文件
        return 1;
    }
    //获取 VideoRender filter
    pWnd->pBase == NULL;
    IBaseFilter *pRenderFiler = NULL;
    //获取  认的视  显示  滤器
    pWnd->pGraph->FindFilterByName(L"Video Renderer",(IBaseFilter**)&pRenderFiler);
    if (pRenderFiler!=NULL)                                    //包含视  信息
    {
        pWnd->m_bViewPlay = TRUE;
        IPin* pVideoIn = NULL;                                 //定义引脚接口指
        //查找  入引脚
        pVideoIn   = pWnd->FindPin(pRenderFiler,PINDIR_INPUT);
```



```
        if (pVideoIn)
        {
            pVideoIn->Disconnect();                          //与视　解码　滤器断开　接
        }
        pWnd->pGraph->RemoveFilter(pRenderFiler);            //移　视　显示　滤器
        //添加 VideoMixingRenderer9　滤器
        pWnd->pGraph->AddFilter(pWnd->pRender,L"Render");
        //获取视　解码器
        pWnd->pGraph->FindFilterByName(L"MPEG Video Decoder",(IBaseFilter**)&pWnd->pBase);
        if (pWnd->pBase)
        {
            IPin* pOutPin = NULL;
            //获取视　解码的　出引脚
            pOutPin = pWnd->FindPin(pWnd->pBase,PINDIR_OUTPUT);
            if (pOutPin != NULL)
            {
                IPin* pColorIn = NULL;
                IPin* pColorOut = NULL;
                //获取　入和　出引脚
                IPin *pTextIn = NULL;
                IPin *pTextOut = NULL;
                IPin *pRenderIn = NULL;
                pRenderIn = pWnd->FindPin(pWnd->pRender,PINDIR_INPUT);
                HRESULT hRet = 0;
                hRet = pOutPin->Disconnect();                //断开　接
                //　接视　解码　滤器　出引脚与 VideoMixingRenderer9　滤器　入引脚
                hRet = pWnd->pGraph->ConnectDirect(pOutPin,pRenderIn,NULL);
            }
        }
    }
    else
    {
        pWnd->m_bViewPlay = FALSE;                           //没有视　信息
    }
    pWnd->m_ThreadStop = TRUE;
    return 0;
}
```

（8）添加一个全局函数 ThreadProc，该函数将作为一个线程函数。当播放一个媒体文件时将创建一个线程来检测媒体文件是否播放完成，如果没有播放完成，则向主窗口发送 CM_POSCHANGE 消息执行 OnPosChange 方法显示当前的播放时间，否则向主窗口发送 CM_COMPLETE 消息执行 Done 方法结束媒体文件播放。实现代码如下：

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
    CDirectShowEventDlg* pWnd = (CDirectShowEventDlg*)lpParameter;    //获取主窗口指
    HANDLE    hEvent;                                                 //定义事件句柄
    pWnd->pEvent->GetEventHandle((OAEVENT*) &hEvent);                 //获取事件句柄
```

```
    long code,p1,p2;                                              //定义事件代码参数
    BOOL    done = FALSE;
    while (!done)                                                 //开始执行循环
    {
        pWnd->SendMessage(CM_POSCHANGE);                          //发    CM_POSCHANGE 消息
        if (WaitForSingleObject(hEvent,80)==WAIT_OBJECT_0)        //Direct Show 是否有事件产生
        {
            //获取事件代码
            while (SUCCEEDED(pWnd->pEvent->GetEvent(&code,&p1,&p2,0)))
            {
                pWnd->pEvent->FreeEventParams(code,p1,p2);        //    放事件参数
                if (code==EC_COMPLETE)                            //是否为播放完成
                {
                    pWnd->m_Completed = TRUE;                     //设置播放完成状态
                    pWnd->SendMessage(CM_COMPLETE);               //发    CM_COMPLETE 消息
                    done=true;                                    //    出循环，结束线程
                }
            }
        }
    }
    return 0;
}
```

（9）处理“打开文件”按钮的单击事件，利用文件打开对话框选择一个媒体文件，然后调用 PlayFile 方法播放媒体文件。相应代码如下：

```
void CDirectShowEventDlg::OnSetFile()
{
    CFileDialog fDlg(TRUE,NULL,NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        "avi 文件|*.avi;*.dat;*.mp3;*.wav;*.mpeg|所有文件|*.*||",this);    //定义文件打开对话框
    if (fDlg.DoModal()==IDOK)
    {
        m_FileName = fDlg.GetPathName();                          //获取文件名
        m_Stop.SendMessage(WM_LBUTTONDOWN,0,0);
        PlayFile(m_FileName);                                     //开始播放媒体文件
    }
}
```

（10）处理“抓图”按钮的单击事件，将抓取当前视频窗口的图像信息，将其保存到磁盘文件中。主要方法是使用 IbasicVideo 接口的 GetCurrentImage 方法来获得位图的信息和位图数据。相应代码如下：

```
void CDirectShowEventDlg::OnSnap()
{
    CFileDialog flDlg(FALSE,"","Snap.bmp");                       //定义文件保存对话框
    if (pGraph != NULL)
    {
        IBasicVideo * pBasicVideo = NULL;                         //定义 IBasicVideo 接口指
        //获取 IBasicVideo 接口对象
        pGraph->QueryInterface(IID_IBasicVideo, (void **)&pBasicVideo);
```



```
        if (pBasicVideo != NULL)
        {
            pMediaControl->Pause();                             //暂停播放
            if (flDlg.DoModal()==IDOK)
            {
                CString csSaveName = flDlg.GetPathName();       //获取文件名
                //获取图像大小，包含位图信息头
                long lBmpSize;
                if (SUCCEEDED(pBasicVideo->GetCurrentImage(&lBmpSize, 0)))
                {
                    //定义图像数据缓冲区，获取图像数据
                    BYTE* pData = new BYTE[lBmpSize];
                    if (SUCCEEDED(pBasicVideo->GetCurrentImage(&lBmpSize, (long*)pData)))
                    {
                        BITMAPFILEHEADER bFile;
                        bFile.bfReserved1 = bFile.bfReserved2 = 0;
                        bFile.bfSize = sizeof(BITMAPFILEHEADER);
                        bFile.bfType = 0x4d42;
                        bFile.bfOffBits = sizeof(BITMAPFILEHEADER)+
                            sizeof(BITMAPINFOHEADER);
                        CFile file;                             //定义文件对象
                        //创建并打开文件
                        file.Open(csSaveName,CFile::modeCreate|CFile::modeReadWrite);
                        //写入文件数据
                        file.Write(&bFile,sizeof(BITMAPFILEHEADER));
                        file.WriteHuge(pData,lBmpSize);
                        file.Close();                           //关  文件
                    }
                    delete [] pData;                            //  放位图数据
                    pBasicVideo->Release();                     //  放 IBasicVideo 接口指
                }
            }
            pMediaControl->Run();                               //继续播放媒体文件
        }
    }
}
```

（11）处理“全屏”按钮的单击事件，调用视频窗口对象的 put_FullScreenMode 方法设置全屏模式。实现代码如下：

```
void CDirectShowEventDlg::OnFullScreen()
{
    if (pViewWnd!= NULL && m_bStop==FALSE)
    {
        pViewWnd->put_FullScreenMode(-1);                       //设置全屏显示
        m_bFullScreen = TRUE;
    }
}
```

（12）改写对话框的 PreTranslateMessage 方法，在全屏模式下按 Esc 键将取消全屏显示。实现代码如下：

```
BOOL CDirectShowEventDlg::PreTranslateMessage(MSG* pMsg)
{
	if (pMsg->message == WM_KEYDOWN)				//是否为按　消息
	{
		if (pMsg->wParam == VK_ESCAPE)				//是否为 Esc
		{
			if (pViewWnd!= NULL)
			{
				pMsg->wParam == 0;
				pViewWnd->put_FullScreenMode(0);	//取消全屏显示
				m_bFullScreen = FALSE;
				return TRUE;
			}
		}
	}
	return CDialog::PreTranslateMessage(pMsg);
}
```

（13）向对话框中添加 SetViewInfo 方法，用于设置视频图像的颜色，如亮度、饱和度、对比度的信息。相应代码如下：

```
void CDirectShowEventDlg::SetViewInfo(int nFlag, float fValue)
{
	IVMRMixerControl9 * pControl = NULL;			//定义 IVMRMixerControl9 接口指
	if (pRender != NULL)
	{
		//获取 IVMRMixerControl9 接口对象
		pRender->QueryInterface(IID_IVMRMixerControl9,(void**)&pControl);
		if (pControl != NULL)
		{
			VMR9ProcAmpControl vmrParam;				//定义参数
			memset(&vmrParam,0,sizeof(VMR9ProcAmpControl));	//初始化参数
			vmrParam.dwSize = sizeof(VMR9ProcAmpControl);	//设置参数大小
			vmrParam.dwFlags = nFlag;				//设置标记
			vmrParam.Brightness = fValue;				//设置亮度值
			vmrParam.Contrast = fValue;				//设置对比度值
			vmrParam.Hue = fValue;					//设置色调值
			vmrParam.Saturation = fValue;				//设置　和度值
			pControl->SetProcAmpControl(0,&vmrParam);		//设置视　图像　色信息
		}
	}
}
```

（14）处理“快进”按钮的单击事件，利用 IMediaPosition 接口的 put_CurrentPosition 方法来设置当前播放的位置，以实现快进功能。实现代码如下：



```
void CDirectShowEventDlg::OnAddSpeed()
{
	IMediaPosition* pPosition = NULL;					//定义 IMediaPosition 接口指
	if (pGraph != NULL)
	{
		pGraph->QueryInterface(IID_IMediaPosition,(void**)&pPosition);	//获取 IMediaPosition 接口对象
		if (pPosition != NULL)
		{
			REFTIME curTime,endTime;
			pPosition->get_StopTime(&endTime);			//获取播放的停止时
			pPosition->get_CurrentPosition(&curTime);		//获取当前的播放时
			curTime += 5;						//设置快
			if (curTime <=endTime)
			{
				pPosition->put_CurrentPosition(curTime);	//设置当前播放的时
			}
			else
			{
				pPosition->put_CurrentPosition(endTime);	//设置播放时　为停止时
			}
		}
	}
}
```

（15）处理“增大音量”按钮的单击事件，利用 IbasicAudi 接口的 put_Volume 方法来调节音量。相应代码如下：

```
void CDirectShowEventDlg::OnVolumnmax()
{
	IBasicAudio* pAudio = NULL;						//定义 IBasicAudio 接口指
	if (pGraph != NULL)
	{
		pGraph->QueryInterface(IID_IBasicAudio,(void**)&pAudio);	//获取 IBasicAudio 接口对象
		if (pAudio != NULL)						//如果有　　数据
		{
			pAudio->get_Volume(&m_lVolumn);				//获取当前的
			if (m_lVolumn <0)					//当前不是最大
			{
				m_lVolumn += 200;				//增加
				pAudio->put_Volume(m_lVolumn);			//设置
			}
			else
			{
				m_lVolumn = 0;					//当前　　为最大
			}
		}
	}
}
```

（16）处理“黑白图像”按钮的单击事件，调节视频图像的饱和度为最小值，这样就实现了黑白图像的效果。实现代码如下：

```
void CDirectShowEventDlg::OnGray()
{
	IVMRMixerControl9 * pControl = NULL;				//定义 IVMRMixerControl9 接口指
	if (pRender != NULL)
	{
		//获取 IVMRMixerControl9 接口对象
		pRender->QueryInterface(IID_IVMRMixerControl9,(void**)&pControl);
		if (pControl != NULL)
		{
			VMR9ProcAmpControl vmrParam;				//定义参数
			memset(&vmrParam,0,sizeof(VMR9ProcAmpControl));	//初始化参数
			vmrParam.dwSize = sizeof(VMR9ProcAmpControl);		//设置参数大小
			vmrParam.dwFlags = ProcAmpControl9_Saturation;		//设置参数标记
			//获取　和度的最小值
			VMR9ProcAmpControlRange range;
			range.dwSize = sizeof(VMR9ProcAmpControlRange);
			range.dwProperty = ProcAmpControl9_Saturation;
			pControl->GetProcAmpControlRange(0, &range);
			//设置　和度为最小值
			if (m_bGrayImage==FALSE)				//设置　白图像
			{
				VMR9ProcAmpControl getParam;			//定义参数
				memset(&getParam,0,sizeof(VMR9ProcAmpControl));	//初始化参数
				getParam.dwSize = sizeof(VMR9ProcAmpControl);	//设置参数大小
				getParam.dwFlags = ProcAmpControl9_Saturation;	//设置参数标记
				pControl->GetProcAmpControl(0,&getParam);	//获取　和度
				m_fSaturation = range.MinValue;
				vmrParam.Saturation = m_fSaturation;		//设置　和度值
				m_bGrayImage = TRUE;
				m_GrayBtn.SetWindowText("彩色图像");
			}
			else						//设置彩色图像
			{
				m_fSaturation = 1;
				m_GrayBtn.SetWindowText("　白图像");
				vmrParam.Saturation = m_fSaturation;
				m_bGrayImage = FALSE;
			}
			//设置　白图像或彩色图像
			HRESULT hRet = pControl->SetProcAmpControl(0,&vmrParam);
		}
	}
}
```



8.6　视频显示窗口设计



8.6.1　视　显示窗口概

视频显示窗口用于显示视频图像画面，在该窗口中用户可以打开新的媒体文件，全屏显示视频图像，设置黑白图像、彩色图像以及停止播放等。视频显示窗口效果如图 8.12 所示。



图 8.12　视频显示窗口

8.6.2　视　显示窗口界　设计

视频显示窗口界面设计过程如下：

（1）创建一个对话框类，类名为 CDisplayWnd。

（2）向对话框中添加一个图片控件，设置图片控件的类型为 Bitmap，设置 Image 属性为程序中导入的位图资源 ID。



8.6.3　视　显示窗口实现　程

视频显示窗口实现过程如下：

（1）向对话框中添加两个菜单对象，用于在鼠标右击对话框时弹出菜单。

```
CIconMenu* m_SubMenu;
CIconMenu m_Menu;
```

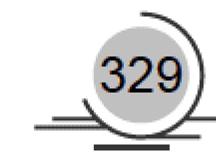
（2）在对话框初始化时加载菜单资源。相应代码如下：

```
BOOL CDisplayWnd::OnInitDialog()
{
	CDialog::OnInitDialog();
	m_Menu.LoadMenu(IDR_VIDEOMENU);                    //加　菜单资源
	m_Menu.ChangeMenuItem(&m_Menu);
	return TRUE;
}
```

（3）处理对话框的 WM_CONTEXTMENU 消息，在鼠标右击对话框时弹出菜单的代码如下：

```
void CDisplayWnd::OnContextMenu(CWnd* pWnd, CPoint point)
{
	CMenu* pMenu = m_Menu.GetSubMenu(0);                //获取子菜单
	//弹出子菜单
	pMenu->TrackPopupMenu(TPM_LEFTBUTTON|TPM_LEFTALIGN,point.x,point.y,this);
}
```

（4）处理对话框的 WM_SIZE 消息，在播放视频文件时，调整对话框大小的同时，调整视频图像的大小使其填充整个窗口。相应代码如下：



```
void CDisplayWnd::OnSize(UINT nType, int cx, int cy)
{
	CDialog::OnSize(nType, cx, cy);
	//获取主窗口指
	CDirectShowEventDlg *pDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
	if (pDlg->m_bViewPlay)
	{
		CRect ClientRC,rc;
		GetClientRect(ClientRC);                         //获取当前窗口的客户区域
		m_Panel.MoveWindow(ClientRC);
		m_Panel.ModifyStyle(SS_BITMAP,SS_BLACKRECT);     //修改  态文本控件  格
		m_Panel.GetClientRect(rc);                       //获取  态文本控件区域
		pDlg->pViewWnd->put_Height(rc.Height());         //设计视  图像的  度
		pDlg->pViewWnd->put_Width(rc.Width());           //设置视  图像的宽度
	}
}
```

（5）处理“播放文件”菜单项的单击事件，调用主窗口中的 OnSetFile 方法来播放视频文件。相应代码如下：

```
void CDisplayWnd::OnPlayfile()
{
	//获取主窗口指
	CDirectShowEventDlg *pDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
	pDlg->OnSetFile();                    //调用主对话框的打开文件按  的单击事件函数播放文件
}
```



8.7　字幕叠加窗口设计

8.7.1　字幕叠加窗口概

字幕叠加窗口用于为视频图像添加或删除字幕信息，在设计字幕信息时用户可以设置字体大小、字体颜色、文本信息和文本的坐标位置等。字幕叠加窗口如图 8.13 所示。

字幕叠加的视频窗口效果如图 8.14 所示。

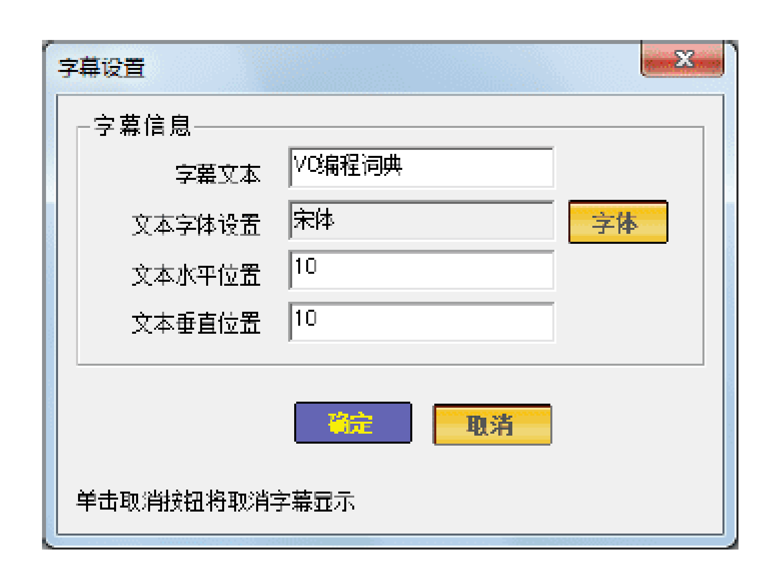


图 8.13　字幕叠加窗口



图 8.14　字幕叠加效果



8.7.2 字幕叠加窗口界 设计

字幕叠加窗口设计过程如下：

（1）新建一个对话框类，类名为 COverlayText。

（2）向对话框中添加按钮、静态文本、群组框等控件。

（3）设置控件属性，如表 8.2 所示。

表 8.2 字幕叠加窗口控件属性设置

控件 ID	控 件 属 性	关 联 变
IDC_FONT	Read only：TRUE	CString：m_Font
IDC_HORPOS	Number：TRUE	UINT：m_HorPos
IDC_VERPOS	Number：TRUE	UINT：m_VerPos
IDC_TEXT	默认	CString：m_Text

8.7.3 字幕叠加窗口实现 程

字幕叠加窗口实现过程如下：

（1）处理“字体”按钮的单击事件，弹出字体设置对话框，设置字体的大小、颜色等信息。相应代码如下：

```
void COverlayText::OnSetFont()
{
	UpdateData();
	CFontDialog ftDlg;                                    //定义字体对话框
	if (ftDlg.DoModal()==IDOK)
	{
		ftDlg.GetCurrentFont(&m_LogFont);                 //获取用户设置的字体
		m_TextColor = ftDlg.GetColor();                   //获取字体 色
		m_Font = ftDlg.GetFaceName();                     //获取字体名称
		UpdateData(FALSE);
	}
}
```

（2）处理“确认”按钮的单击事件，关闭对话框的相应代码如下。

```
void COverlayText::OnConfirm()
{
	EndDialog(IDOK);                                      //关 对话框
}
```

（3）处理“取消”按钮的单击事件，取消字幕信息，关闭对话框。实现代码如下：

```
void COverlayText::OnCancel()
{
```

```
        m_Text = " ";
        UpdateData(FALSE);
        this->GetFont()->GetLogFont(&m_LogFont);                //获取   认的字体
        EndDialog(IDOK);                                        //关   对话框
}
```

在字幕叠加窗口模块并没有实现字幕叠加和取消字幕叠加的功能，它只是提供字幕信息，实现字幕叠加和取消字幕叠加是在主窗口中实现的。下面给出主窗口中实现字幕叠加的相关代码。

```
void CDirectShowEventDlg::OnOverlayText()
{
        if (pRender != NULL)
        {
                IVMRMixerBitmap9 * pBmp9 = NULL;                //定义 IVMRMixerBitmap9 接口指
                //获取 IVMRMixerBitmap9 接口对象
                pRender->QueryInterface(IID_IVMRMixerBitmap9,(void**)&pBmp9);
                if (pBmp9!= NULL)
                {
                        COverlayText OverlayDlg;                //定义字幕叠加对话框
                        if (OverlayDlg.DoModal()==IDOK)         //显示字幕叠加对话框
                        {
                                BYTE byR,byG,byB;               //定义   色变
                                byR = GetRValue(OverlayDlg.m_TextColor);   //获取文本   色
                                byG = GetGValue(OverlayDlg.m_TextColor);
                                byB = GetBValue(OverlayDlg.m_TextColor);
                                LOGFONT logfont = OverlayDlg.m_LogFont;    //获取字体信息
                                int nX = OverlayDlg.m_HorPos;              //获取坐标
                                int nY = OverlayDlg.m_VerPos;
                                IVideoWindow * pVideoWnd = NULL;           //定义 IVideoWindow 接口指
                                long lVideoWidth, lVideoHeight;
                                //获取 IVideoWindow 接口指
                                pRender->QueryInterface(IID_IVideoWindow, (void**)&pVideoWnd);
                                if (pVideoWnd!= NULL)
                                {
                                        pVideoWnd->get_Width(&lVideoWidth);    //获取视   窗口的宽度
                                        pVideoWnd->get_Height(&lVideoHeight);  //获取视   窗口的   度
                                        //创建一个设备上下文
                                        HDC hBmpDC = CreateCompatibleDC(GetDC()->m_hDC);
                                        CFont Font;                            //定义字体对象
                                        Font.CreateFontIndirect(&logfont);     //创建字体
                                        //   中字体对象
                                        HFONT hOldFont = (HFONT) SelectObject(hBmpDC,Font.m_hObject);
                                        int nLength, nTextBmpWidth, nTextBmpHeight;
                                        SIZE szText={0};
                                        nLength = strlen(OverlayDlg.m_Text);   //获取字符串的   度
                                        //获取文本的   度和   度
                                        GetTextExtentPoint32(hBmpDC, OverlayDlg.m_Text, nLength, &szText);
```

```
                    nTextBmpHeight = szText.cy;
                    nTextBmpWidth   = szText.cx;
                    HBITMAP hBmp = CreateCompatibleBitmap(GetDC()->m_hDC,
                    nTextBmpWidth, nTextBmpHeight);                    //创建位图
                    BITMAP bmObj;                                      //定义位图信息对象
                    HBITMAP hbmOld;                                    //定义位图句柄
                    GetObject(hBmp, sizeof(bmObj), &bmObj);            //获取位图信息
                    hbmOld = (HBITMAP)SelectObject(hBmpDC, hBmp);      //   中位图
                    //定义文本的矩形区域
                    RECT rcText;
                    SetRect(&rcText, 0, 0, nTextBmpWidth, nTextBmpHeight);
                    //设置背景  色，注意与文本  色接  ，否则效果不好
                    SetBkColor(hBmpDC, RGB(byR, byG, byB-1));
                    SetTextColor(hBmpDC, RGB(byR, byG, byB));          //设置文本  色
                    TextOut(hBmpDC, 0, 0,OverlayDlg.m_Text, nLength);  //   出文本
                    VMR9AlphaBitmap bmpInfo;                           //定义参数
                    ZeroMemory(&bmpInfo, sizeof(bmpInfo));             //初始化参数
                    bmpInfo.dwFlags = VMRBITMAP_HDC;                   //设置参数标记
                    bmpInfo.hdc = hBmpDC;                              //设置设备上下文
                    //等比例  出文字
                    double xRate = (double)nTextBmpWidth /lVideoWidth;
                    double yRate = (double)nTextBmpHeight / lVideoHeight;
                    double fX = (double)nX / lVideoWidth;              //确定文本  出比例
                    double fY = (double)nY / lVideoHeight;
                    bmpInfo.rDest.left   = fX;
                    bmpInfo.rDest.right = fX+xRate;
                    bmpInfo.rDest.top = fY;
                    bmpInfo.rDest.bottom = fY+ yRate;
                    bmpInfo.rSrc = rcText;                             //设置文本显示区域
                    bmpInfo.clrSrcKey = RGB(byR, byG, byB-1);          //设置关    色
                    bmpInfo.dwFlags |= VMRBITMAP_SRCCOLORKEY;
                    bmpInfo.fAlpha = 1.0;
                    pBmp9->SetAlphaBitmap(&bmpInfo);                   //实现图像字幕叠加
                }
            }
        }
    }
}
```



8.8　视频设置窗口设计

8.8.1　视　设置窗口概

视频设置窗口主要用于设置视频图像的色彩信息，包括视频图像的亮度、饱和度和对比度等。视



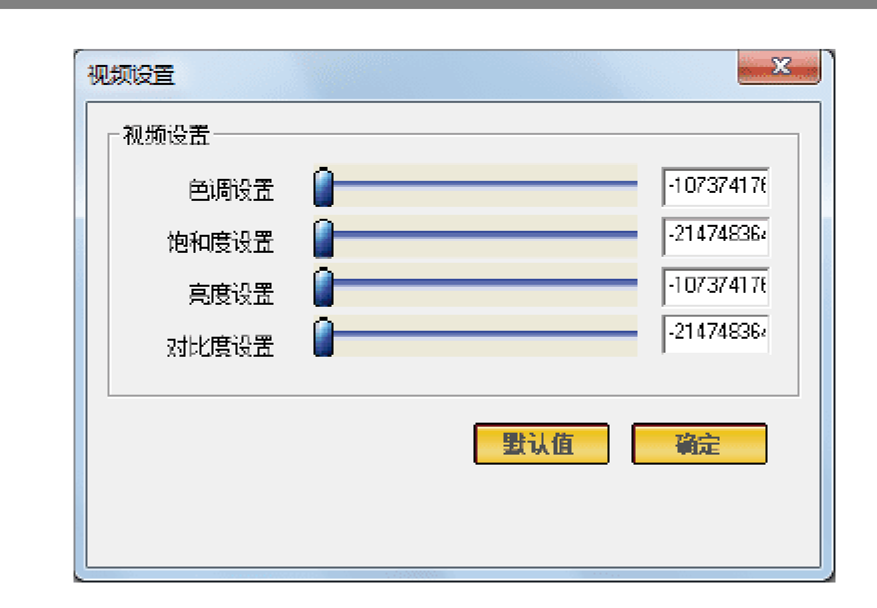


图 8.15　视频设置窗口

频设置窗口如图 8.15 所示。

8.8.2　视　设置窗口界　设计

视频设置窗口设计过程如下：

（1）新建一个对话框类，类名为 CVideoSet。

（2）向对话框中添加按钮、静态文本、群组框、编辑框、滑块等控件。

（3）设置控件属性，如表 8.3 所示。

表 8.3　视　设置窗口控件属性设置

控件 ID	控 件 属 性	关 联 变
IDC_HUENUM	默认	CEdit：m_HueNum
IDC_HUE	Auto ticks：TRUE	CCustomSlider
IDC_CONTRASTNUM	默认	CEdit：m_ConNum
IDC_CONTRAST	Auto ticks：TRUE	CCustomSlider：m_Contrast
IDC_DEFAULT	Caption：默认值	无

8.8.3　视　设置窗口实现　程

（1）处理对话框的 WM_HSCROLL 消息，在用户拖动滑块时，将执行该消息处理函数，设置视频图像相应的信息。实现代码如下：

```
void CVideoSet::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	if (pScrollBar != NULL)
	{
		CDirectShowEventDlg *pMainDlg = NULL;
		//获取对话框主窗口指
		pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
		pMainDlg->m_bGrayImage = FALSE;
		pMainDlg->m_GrayBtn.SetWindowText("   白图像");		//设置按   文本
		pMainDlg->m_GrayBtn.Invalidate();				//更新按
		if (pScrollBar->m_hWnd==m_Hue.m_hWnd)			//色调滑块的滚动消息
		{
			if (nSBCode==SB_THUMBPOSITION)			//拖动滚动块
			{
				m_Hue.SetPos(nPos);				//设置滑块位置
			}
			int nCurPos = m_Hue.GetPos();			//获取滑块位置
			CString csPos;
			csPos.Format("%i",nCurPos);
			m_HueNum.SetWindowText(csPos);
```

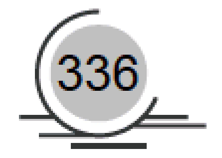


```
            //调用主对话框的 SetViewInfo 方法设置视　图像的色调
            pMainDlg->SetViewInfo(ProcAmpControl9_Hue,nCurPos);
            pMainDlg->m_fHue = nCurPos;
        }
        else if (pScrollBar->m_hWnd==m_Saturation.m_hWnd)          //　和度滑块的滚动消息
        {
            if (nSBCode==SB_THUMBPOSITION)                         //拖动滚动块
            {
                m_Saturation.SetPos(nPos);                         //设置滑块位置
            }
            int nCurPos = m_Saturation.GetPos();                   //获取滑块位置
            CString csPos;
            csPos.Format("%i",nCurPos);
            m_SatNum.SetWindowText(csPos);
            //调用主对话框的 SetViewInfo 方法设置视　图像的　和度
            pMainDlg->SetViewInfo(ProcAmpControl9_Saturation,nCurPos/100.0);
            pMainDlg->m_fSaturation = nCurPos/100.0;
        }
        else if (pScrollBar->m_hWnd==m_Brightness.m_hWnd)          //亮度滑块的滚动消息
        {
            if (nSBCode==SB_THUMBPOSITION)                         //拖动滚动块
            {
                m_Brightness.SetPos(nPos);                         //设置滑块位置
            }
            int nCurPos = m_Brightness.GetPos();                   //获取滑块位置
            CString csPos;
            csPos.Format("%i",nCurPos);
            m_BrightNum.SetWindowText(csPos);
            //调用主对话框的 SetViewInfo 方法设置视　图像的亮度
            pMainDlg->SetViewInfo(ProcAmpControl9_Brightness,nCurPos);
            pMainDlg->m_fBright = nCurPos;
        }
        else if (pScrollBar->m_hWnd==m_Contrast.m_hWnd)            //对比度滑块的滚动消息
        {
            if (nSBCode==SB_THUMBPOSITION)                         //拖动滚动块
            {
                m_Contrast.SetPos(nPos);                           //设置滑块位置
            }
            int nCurPos = m_Contrast.GetPos();                     //获取滑块位置
            CString csPos;
            csPos.Format("%i",nCurPos);
            m_ConNum.SetWindowText(csPos);
            //调用主对话框的 SetViewInfo 方法设置视　图像的对比度
            pMainDlg->SetViewInfo(ProcAmpControl9_Contrast,nCurPos/100.0);
            pMainDlg->m_fContrast = nCurPos/100.0;
        }
    }
    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

（2）在对话框初始化时获取当前视频图像各项信息的默认值，作为视频设置窗口的初始值。相应代码如下：

```
BOOL CVideoSet::OnInitDialog()
{
	CDialog::OnInitDialog();
	CDirectShowEventDlg *pMainDlg = NULL;
	//获取对话框主窗口指
	pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
	if (pMainDlg)
	{
		//设置色调滑块的滚动范围
			m_Hue.SetRange(pMainDlg->m_HueRange.MinValue,pMainDlg->m_HueRange.MaxValue);
		WPARAM wParam;
		MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fHue);
		m_Hue.SetPos(100);
		m_Hue.SetPos(pMainDlg->m_fHue);
		//执行对话框的 WM_HSCROLL 消息处理函数
		SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Hue.m_hWnd);
		//设置  和度滑块的滚动范围
		m_Saturation.SetRange(pMainDlg->m_SatRange.MinValue*100,
			pMainDlg->m_SatRange.MaxValue*100);
		MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fSaturation*100);
		m_Saturation.SetPos(100);
		m_Saturation.SetPos(pMainDlg->m_fSaturation*100);
		//执行对话框的 WM_HSCROLL 消息处理函数
		SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Saturation.m_hWnd);
		//设置亮度滑块的滚动范围
		m_Brightness.SetRange(pMainDlg->m_BrightRange.MinValue,
			pMainDlg->m_BrightRange.MaxValue);
		MAKEWPARAM(SB_THUMBPOSITION,pMainDlg->m_fBright);
		m_Brightness.SetPos(100);
		m_Brightness.SetPos(pMainDlg->m_fBright);
		//执行对话框的 WM_HSCROLL 消息处理函数
		SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Brightness.m_hWnd);
		//设置对比度滑块的滚动范围
		m_Contrast.SetRange(pMainDlg->m_ConRange.MinValue*100,
			pMainDlg->m_ConRange.MaxValue*100);
		MAKEWPARAM(SB_THUMBPOSITION,1*100);
		m_Contrast.SetPos(100);
		m_Contrast.SetPos(1*100);
		//执行对话框的 WM_HSCROLL 消息处理函数
		SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Contrast.m_hWnd);
	}
	return TRUE;
}
```

（3）处理“默认”按钮的单击事件，恢复视频图像的默认效果。实现代码如下：



```
void CVideoSet::OnDefault()
{
    CDirectShowEventDlg *pMainDlg = NULL;
    pMainDlg = (CDirectShowEventDlg *)AfxGetMainWnd();          //获取主对话框指
    if (pMainDlg)
    {
        //设置色调滑块的滚动范围
        m_Hue.SetRange(pMainDlg->m_HueRange.MinValue,pMainDlg->m_HueRange.MaxValue);
    WPARAM wParam;
        MAKEWPARAM(SB_THUMBPOSITION,1);
        m_Hue.SetPos(100);
        m_Hue.SetPos(1);
        //执行对话框的 WM_HSCROLL 消息处理函数
        SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Hue.m_hWnd);
        pMainDlg->m_fHue = 1;
        //设置　和度滑块的滚动范围
        m_Saturation.SetRange(pMainDlg->m_SatRange.MinValue*100,
            pMainDlg->m_SatRange.MaxValue*100);
        MAKEWPARAM(SB_THUMBPOSITION,1*100);
        m_Saturation.SetPos(100);
        m_Saturation.SetPos(1*100);
        //执行对话框的 WM_HSCROLL 消息处理函数
        SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Saturation.m_hWnd);
        pMainDlg->m_fSaturation = 1;
        //设置亮度滑块的滚动范围
        m_Brightness.SetRange(pMainDlg->m_BrightRange.MinValue,
            pMainDlg->m_BrightRange.MaxValue);
        MAKEWPARAM(SB_THUMBPOSITION,1);
        m_Brightness.SetPos(100);
        m_Brightness.SetPos(1);
        //执行对话框的 WM_HSCROLL 消息处理函数
        SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Brightness.m_hWnd);
        pMainDlg->m_fBright = 1;
        //设置对比度滑块的滚动范围
        m_Contrast.SetRange(pMainDlg->m_ConRange.MinValue*100,
            pMainDlg->m_ConRange.MaxValue*100);
        MAKEWPARAM(SB_THUMBPOSITION,1*100);
        m_Contrast.SetPos(100);
        m_Contrast.SetPos(1*100);
        //执行对话框的 WM_HSCROLL 消息处理函数
        SendMessage(WM_HSCROLL,wParam,(LPARAM)m_Contrast.m_hWnd);
        pMainDlg->m_fContrast = 1;
    }
}
```

（4）处理“色调”编辑框的文本改变时的事件，当用户在编辑框中输入色调值时将设置滑块显示的位置，这样会触发对话框的 WM_HSCROLL 消息，最终在 WM_HSCROLL 消息处理函数中设置色调信息。相应代码如下：

```
void CVideoSet::OnChangeHuenum()
{
        CString csText;
        m_HueNum.GetWindowText(csText);                    //获取色调值
        if (!csText.IsEmpty())
        {
                m_Hue.SetPos(atoi(csText));                //设置色调滑块位置
        }
}
```


视频讲解

8.9　文件播放列表窗口设计

8.9.1　文件播放列表窗口概

文件播放列表的主要功能是能够播放一组媒体文件，在一组媒体文件中，当播放完一个文件之后，会随机或按顺序播放下一个文件。用户可以将一组媒体文件以列表形式保存到磁盘文件中，这样可以通过载入列表功能将文件列表添加到播放列表中。文件播放列表窗口如图 8.16 所示。

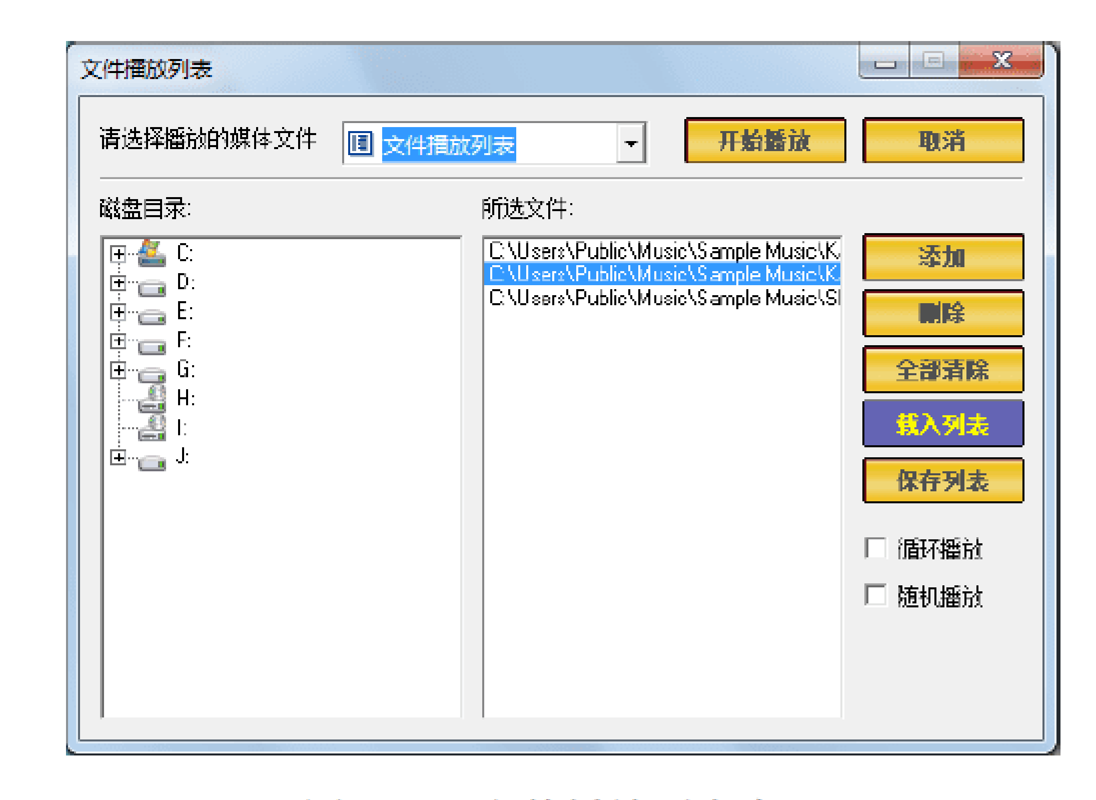


图 8.16　文件播放列表窗口

8.9.2　文件播放列表窗口界　设计

文件播放列表窗口设计过程如下：

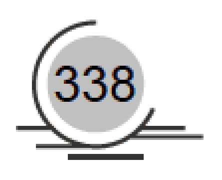
（1）创建一个对话框类，类名为 CFileList。

（2）向对话框中添加静态文本、按钮、列表框、树视图、复选框等控件。

（3）设置主要控件属性，如表 8.4 所示。

表 8.4　文件播放列表窗口属性设置

控件 ID	控 件 属 性	关 联 变
IDC_TREELIST	Has buttons：TRUE Has Lines：TRUE Lines as root：TRUE	CSysTreeCtrl：m_DirList
IDC_LIST	Sort：FALSE	CListBox：m_FileList
IDC_ADD_FILE	Caption：添加 Disable：TRUE	CButton：m_AddFile
IDC_LOOPCHECK	Cation：循环播放 Auto：TRUE	CButton　m_LoopCheck



8.9.3　文件播放列表窗口实现　程

文件播放列表窗口实现过程如下：

（1）处理树节点选中状态改变时的事件，判断当前节点是目录还是文件，如果为文件，“添加”按钮可用，否则“添加”按钮不可用。相应代码如下：

```
void CFileList::OnSelchangedTreelist(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
	HTREEITEM hSelItem = m_DirList.GetSelectedItem();		//获取　中的
	if (hSelItem)							//有　　被　中
	{
		CString csPath = m_DirList.GetFullPath(hSelItem);	//获取文件的完整名称
		CFileFind flFind;
		BOOL bFind = flFind.FindFile(csPath);			//查找文件
		m_AddFile.EnableWindow(FALSE);				//禁用“添加”按
		if (bFind)
		{
			flFind.FindNextFile();				//查找下一个文件
			if (!flFind.IsDirectory())			//如果不是目录
			{
				m_AddFile.EnableWindow();		//激活“添加”按
			}
		}
		m_AddFile.Invalidate();					//更新“添加”按
	}
	*pResult = 0;
}
```

（2）处理双击树节点时的事件，判断当前节点是目录还是文件，如果为文件，将当前文件添加到列表中。相应代码如下：

```
//处理树视图的双击事件
void CFileList::OnDblclkTreelist(NMHDR* pNMHDR, LRESULT* pResult)
{
	HTREEITEM hSelItem = m_DirList.GetSelectedItem();		//获取　中的
	if (hSelItem)
	{
		CString csPath = m_DirList.GetFullPath(hSelItem);	//获取文件名
		CFileFind flFind;
		BOOL bFind = flFind.FindFile(csPath);			//查找文件
		if (bFind)
		{
			flFind.FindNextFile();				//查找下一个文件
			if (!flFind.IsDirectory())			//如果不是目录
			{
				m_FileList.AddString(csPath);		//将文件添加到列表中
```

```
                }
            }
        }
        *pResult = 0;
}
```

（3）处理“删除”按钮的单击事件，删除文件列表中的指定文件。实现代码如下：

```
void CFileList::OnDeleteFile()
{
        int nCurSel = m_FileList.GetCurSel();                    //获取当前  中的  目
        if (nCurSel != -1)
        {
                m_FileList.DeleteString(nCurSel);                 //删  当前  中的  目
                if (nCurSel != 0)
                {
                        m_FileList.SetCurSel(nCurSel-1);          //将下一个  目设置为当前  中的  目
                }
        }
}
```

（4）处理“载入列表”按钮的单击事件，加载一组媒体文件到列表中。实现代码如下：

```
void CFileList::OnImportList()
{
        CFileDialog flDlg(TRUE,"","",OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
                "文件列表|*.lst|所有文件|*.*||");                  //定义文件打开对话框
        if (flDlg.DoModal()==IDOK)
        {
                CString csFileName = flDlg.GetPathName();         //获取加  的文件名
                //获取媒体文件数
                int nCount = GetPrivateProfileInt("FileOperation","FileCount",0,csFileName);
                if (nCount > 0)
                {
                        m_FileList.ResetContent();                //删  文件列表中的所有
                        for(int i=0; i<nCount; i++)               //利用循环加  文件到列表中
                        {
                                char csList[MAX_PATH] = {0};
                                CString pchSection;
                                pchSection.Format("FileList%i",i);
                                //读取文件名
                        GetPrivateProfileString("FileList",pchSection,"",csList,MAX_PATH,csFileName);
                                if (strcmp(csList,"")!=0)
                                {
                                        m_FileList.AddString(csList);     //将文件名添加到文件列表中
                                }
                        }
                }
        }
}
```



（5）处理“保存列表”按钮的单击事件，保存文件列表中的媒体文件到磁盘文件中。相应代码如下：

```
void CFileList::OnSaveList()
{
	int nItemCount = m_FileList.GetCount();                    //获取列表    数
	if (nItemCount > 0)
	{
		CFileDialog flDlg(FALSE,"","MediaList.lst");          //定义文件保存对话框
		if (flDlg.DoModal()==IDOK)
		{
			CString csFileName = flDlg.GetPathName();       //获取文件名
			CString csCount;
			csCount.Format("%d",nItemCount);                //获取    目数
			CString csList;
			DeleteFile(csFileName);                          //  先删   文件
			//写入数
			WritePrivateProfileString("FileOperation","FileCount",csCount,csFileName);
			for(int i=0; i<nItemCount; i++)                  //利用循环写入各个文件名
			{
				char chCount[10] = {0};
				CString pchSection;
				pchSection.Format("FileList%i",i);
				m_FileList.GetText(i,csList);                //获取当前      文件名
				//将文件名写入 INI 文件中
				WritePrivateProfileString("FileList",pchSection,csList,csFileName);
				pchSection.ReleaseBuffer();
			}
		}
	}
	else
	{
		MessageBox("文件列表为空","提示");
	}
}
```

（6）处理“开始播放”按钮的单击事件，播放文件列表中的媒体文件。实现代码如下：

```
void CFileList::OnPlayList()
{
	int nCount = m_FileList.GetCount();                        //获取列表    数
	if (nCount>0)
	{
		//获取主对话框指
		CDirectShowEventDlg *pDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
		CString csName;
		m_bStop = FALSE;
		m_bStopComplete = FALSE;
		for(int i=0; i<nCount; i++)
		{
```

```
                if (m_bRandPlay)                                   //如果为  机播放
                {
                    i = rand() % nCount;
                }

                m_FileList.GetText(i,csName);                      //获取播放的媒体文件名
                m_bPlayList = TRUE;
                pDlg->PlayFile(csName);                            //播放媒体文件
                m_FileList.SetCurSel(i);                           //  中当前的文件
                if (m_bStop)
                {
                    pDlg->Done(0,0);                               //停止播放列表
                    m_bStopComplete = TRUE;
                    break;
                }
                while(pDlg->m_Completed==FALSE)
                {
                    MSG msg;
                    GetMessage(&msg,0,0,WM_USER);                  //在循环  程中响应界  操作
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
                    if (m_bStop)
                    {
                        pDlg->Done(0,0);                           //停止播放列表
                        m_bStopComplete = TRUE;
                        return;
                    }
                }
                if (m_bLoopPlay==TRUE || m_bRandPlay==TRUE)        //循环播放
                {
                    if (i==nCount-1)
                    {
                        i = -1;
                    }
                }
            }
        }
}
```

（7）处理文件列表控件双击时的事件，将播放当前选项中的媒体文件。实现代码如下：

```
void CFileList::OnDblclkList()
{
    int nCount = m_FileList.GetCount();                            //获取列表  数
    int nCurPos = m_FileList.GetCurSel();                          //获取当前列表  索引
    if (nCurPos != -1 && nCount>0)
    {
        //获取主对话框指
        CDirectShowEventDlg *pDlg = (CDirectShowEventDlg *)AfxGetMainWnd();
```



```
        CString csName;
        m_bStop = FALSE;
        m_bStopComplete = FALSE;
        pDlg->Invalidate();                                     //更新主对话框
        int i = nCurPos;
        for(; i<nCount; i++)
        {
            m_FileList.GetText(i,csName);                       //获取媒体文件名
            m_bPlayList = TRUE;
            pDlg->PlayFile(csName);                             //开始播放文件
            m_FileList.SetCurSel(i);                            //在列表中　中
            if (m_bStop)
            {
                pDlg->Done(0,0);                                //停止播放列表
                m_bStopComplete = TRUE;
                break;
            }
            while(pDlg->m_Completed==FALSE)
            {
                MSG msg;
                GetMessage(&msg,0,0,WM_USER);                   //在循环　程中响应界　操作
                TranslateMessage(&msg);
                DispatchMessage(&msg);
                if (m_bStop)
                {
                    pDlg->Done(0,0);                            //停止播放列表
                    m_bStopComplete = TRUE;
                    return;
                }
            }
            if (m_bRandPlay)                                    //如果为　机播放
            {
                i = rand() % nCount;
            }
            if (m_bLoopPlay==TRUE || m_bRandPlay==TRUE)         //循环播放
            {
                if (i==nCount-1)
                {
                    i = -1;
                }
            }
        }
    }
}
```

（8）处理“循环播放”复选框的单击事件，如果复选框被选中，将循环播放列表中的文件。实现代码如下：

```
void CFileList::OnLoopcheck()
{
```

```
    int nState = m_LoopCheck.GetCheck();                    //获取复   框的状态
    if (nState)                                             //是否为   中状态
    {
        m_bLoopPlay = TRUE;                                 //表示循环播放
    }
    else
    {
        m_bLoopPlay = FALSE;
    }
}
```

8.10 项目文件清单

飓风影音的项目文件清单如表 8.5 所示。

表 8.5　目文件清单

文件名称	文件类型	文件描	文件名称	文件类型	文件描
CustomGroup.cpp	源文件	制定分类	NumLabel.cpp	源文件	编号
CustomSlider.cpp	源文件	制作幻灯片	OverlayText.cpp	源文件	字幕文件
DirectShowEvent.cpp	源文件	影音播放文件	IconMenu1.cpp	源文件	图标菜单
DirectShowEvent.rc	资源文件	影音播放资源文件	SysTreeCtrl.cpp	源文件	列表控制
DirectShowEventDlg.cpp	源文件	媒体播放	VideoSet.cpp	源文件	视频设置
DisplayDialog.cpp	源文件	视频显示	IconComboBox.cpp	源文件	图标组合框
FileList.cpp	源文件	文件播放列表			

8.11 本章总结

飓风影音设计的媒体播放器采用的是 Direct Show 实现的。使用 Direct Show 不仅可以控制播放进度、音量大小、全屏切换，还可以实现字幕叠加、图像颜色控制等，对于额外的特色功能，用户可以通过设计自己的过滤器来实现。希望对读者在以后的开发中有所帮助。



第 9 章

吃豆子游戏

（Visual Studio 2017 实现）

吃豆子游戏的英文名称为 PacMan，可在多种系统、平台上运行，是一款非常有名的动作休闲游戏。本章将使用 Windows API，依照面向对象的设计方法，逐步完成一个让人爱不释手的迷你游戏。

通过学习本章，读者可以学到：

- 了解 Windows 消息循环的工作原理
- 掌握如何建立一个 Windows 窗口应用程序
- 掌握父类与子类设计
- 掌握少量的 GDI 函数
- 了解函数模板和动态分配的实际用途



9.1 开 发 背 景

随着社会的发展，科技水平的不断提高，网络游戏已经成为人们日常娱乐的一部分。大多数的游戏都具备传染性与普适性，不受地域限制。一个高质量的游戏就像一个火种，将会点燃游戏玩家的激情，并迅速蔓延，占据整个需求市场。

9.2 需 求 分 析

好的游戏会令玩家爱不释手，因此设计与开发出高质量的游戏才能满足广大游戏爱好者的需求，所以本章设计了一个吃豆子游戏，主要面向青少年和老人，用于开发智力，消遣娱乐。



9.3 系 统 设 计

9.3.1 系统目标

本章中设计了一款吃豆子游戏（PacMan），此游戏将具有以下特点。

☑ 规则简单。

☑ 运行速度快。

☑ 灵活性强。

☑ 容易上手。

玩家可以操作的角色是一张“大嘴”，当它避开所有障碍物吃掉所有的豆子时则闯关成功，否则闯关失败。

9.3.2 系统 览

游戏的运行效果如图 9.1～图 9.3 所示。

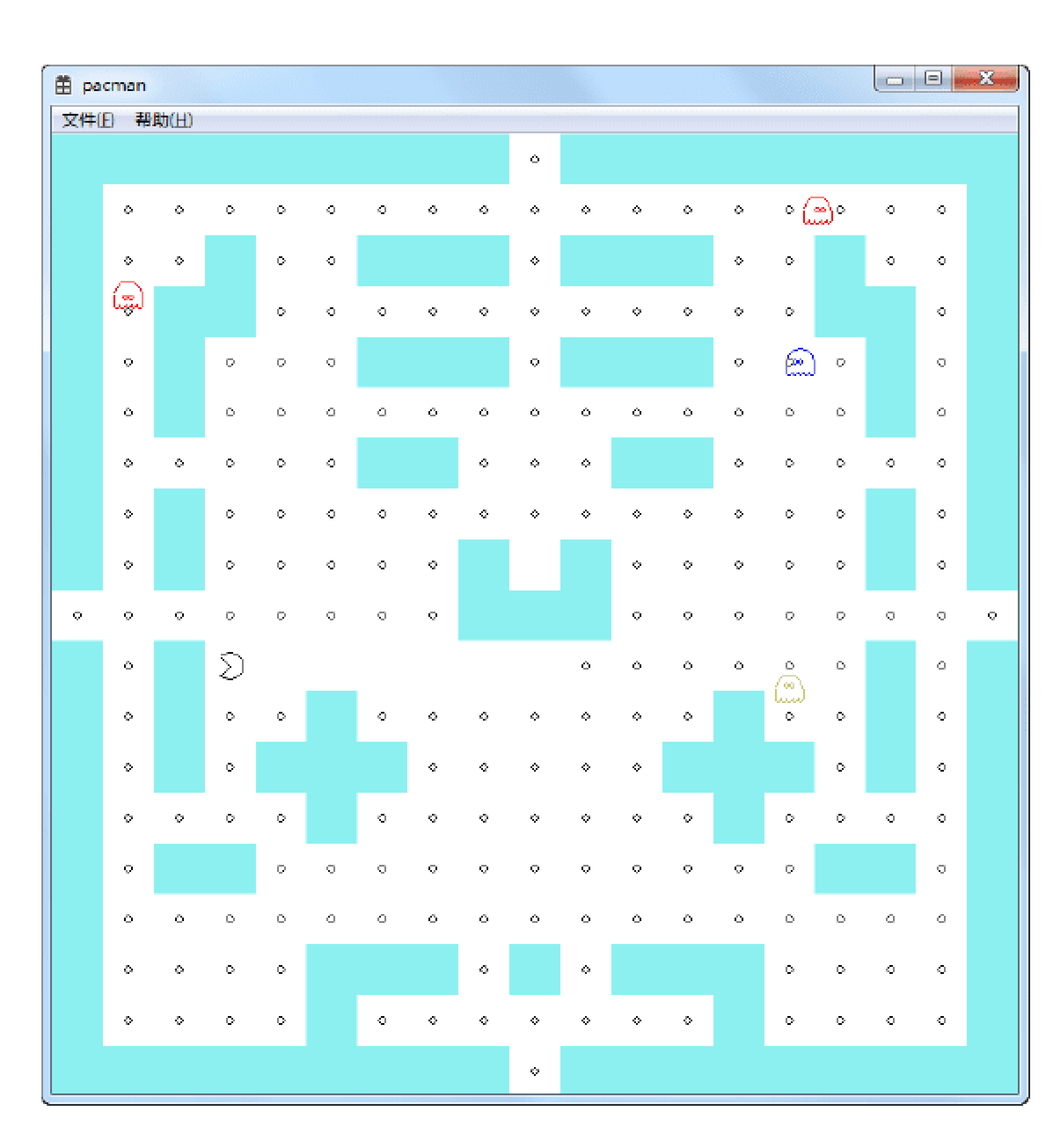


图 9.1 吃豆子游戏第 1 关

9.3.3 业务流程图

吃豆子游戏的业务流程如图 9.4 所示。



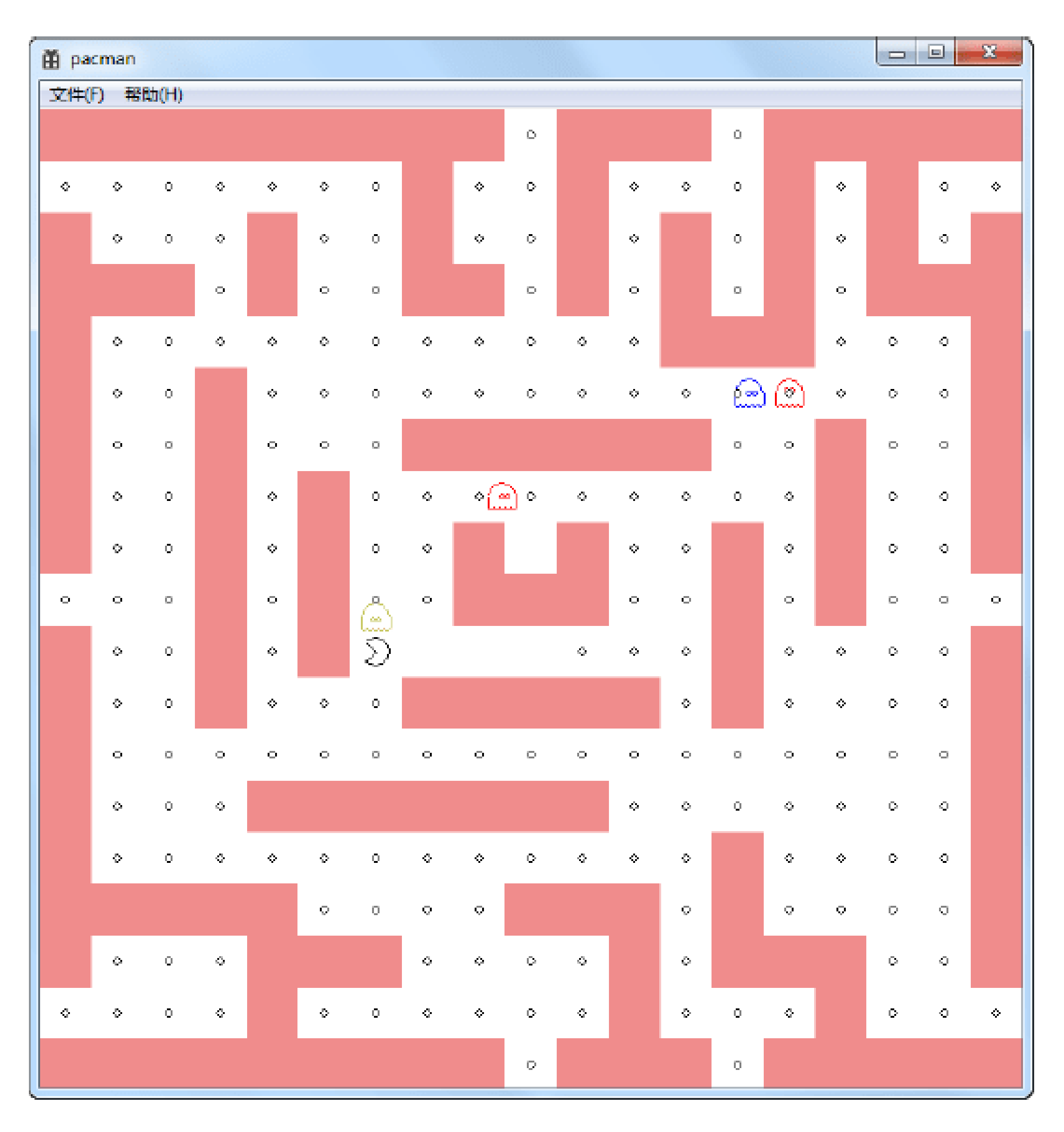


图 9.2　吃豆子游戏第 2 关

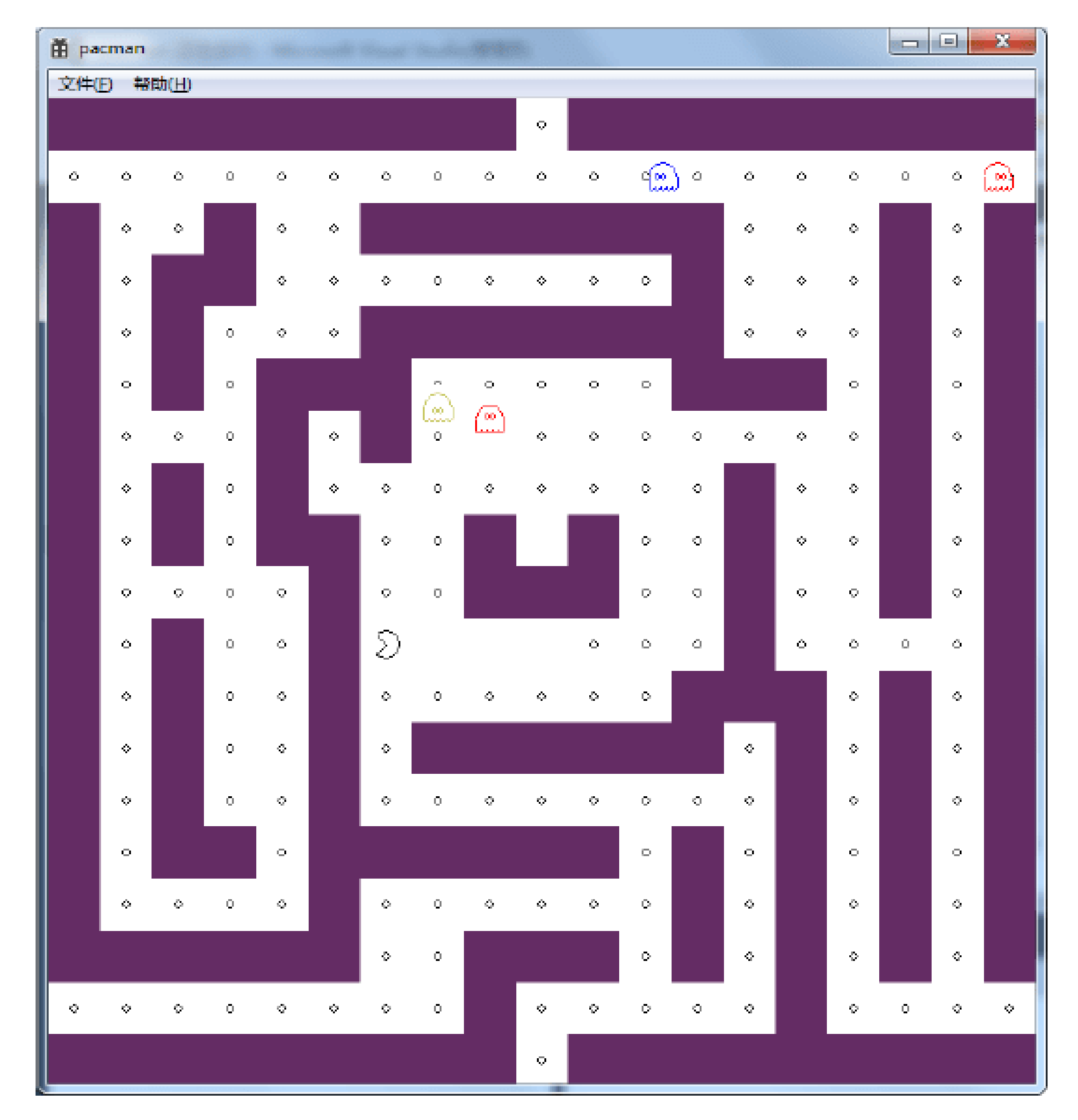


图 9.3　吃豆子游戏第 3 关

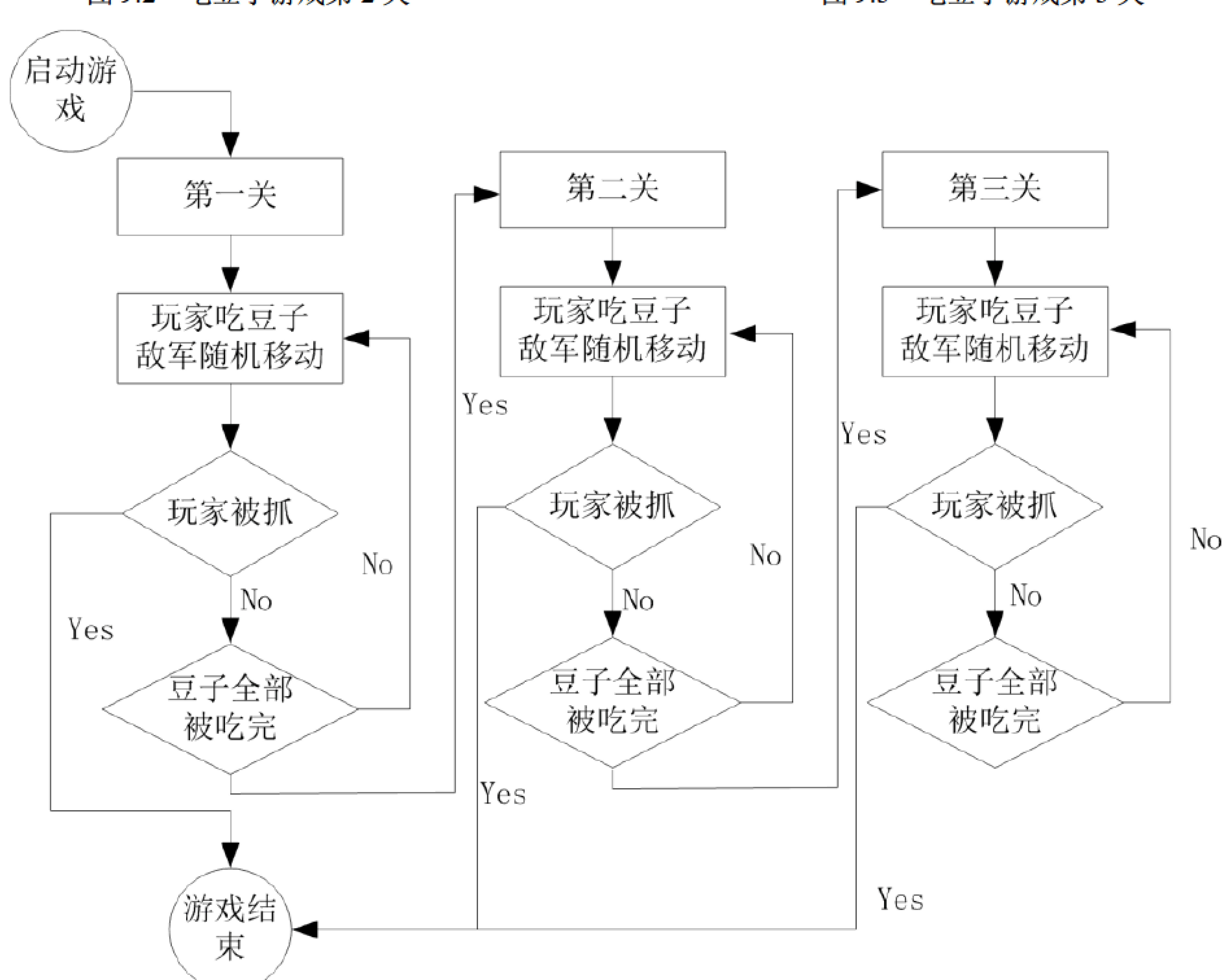


图 9.4　业务流程图





9.4 技 术 分 析

在开始项目之前，读者需要对将要用到的技术和技巧有一定的掌握，以便更好地理解和学习本项目。

9.4.1 建立 Windows 窗口应用程序

使用 Visual Studio 2017 创建 Win32 窗口程序的步骤如下：

（1）打开 Visual Stuido 2017，选择菜单项“文件”→“新建”→“项目”命令，如图 9.5 所示，弹出新建窗口。



图 9.5 新建项目

（2）在弹出的窗口中依次选择 Visual C++→Win32→Win32 项目，在“名称”文本框中输入项目名称 pacman，最后单击“确定”按钮，如图 9.6 所示。

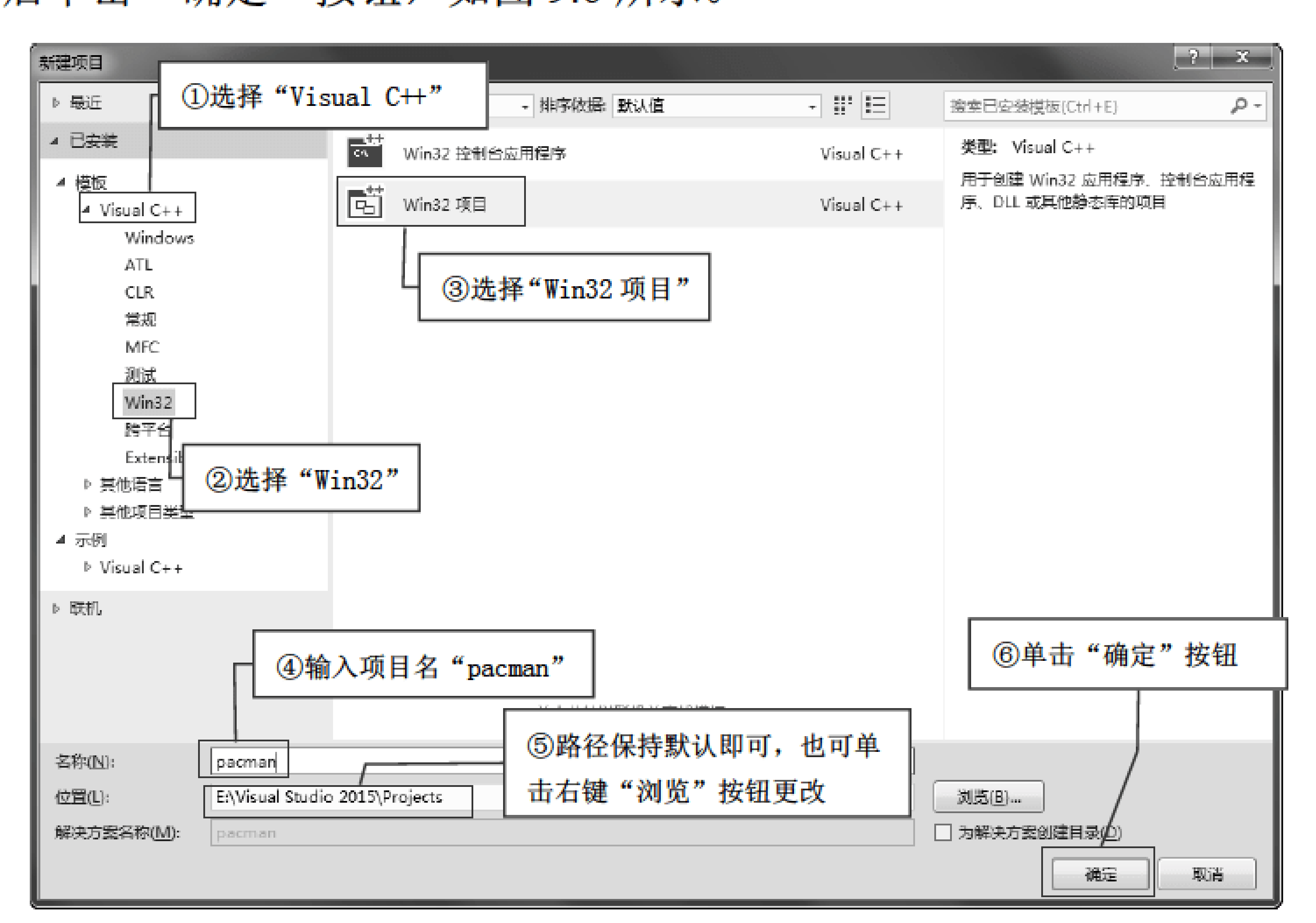


图 9.6 新建项目之输入项目名称



（3）在弹出的“Win32 应用程序向导”对话框中直接单击“下一步”按钮，如图 9.7 所示。



图 9.7　“Win32 应用程序向导”对话框

（4）选中“Windows 应用程序”单选按钮，然后单击“完成”按钮，如图 9.8 所示。

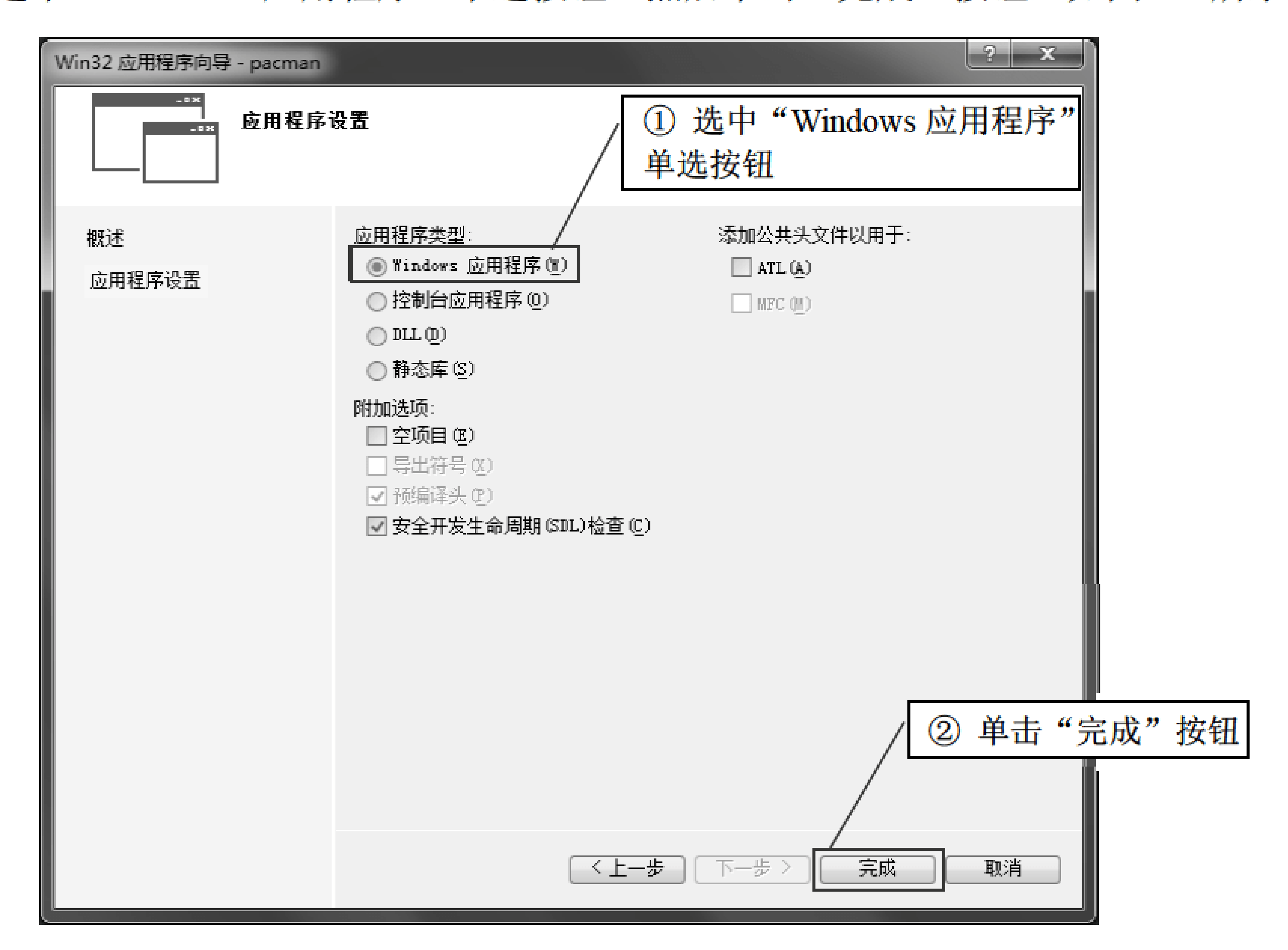


图 9.8　应用程序设置

（5）进行上述操作后生成的项目如图 9.9 所示。

图 9.9 是进行以上操作后生成的程序文件。这是一个只使用 Win32API，没使用其他框架（如 MFC）的 Windows 窗口程序。

至此，我们尚未输入一行代码，IDE（集成开发工具，这里用的是 Visual Studio 2017）已经生成了一个窗口程序。接下来会在此程序的基础上进行调整，逐步实现快乐吃豆子游戏程序。

按照图 9.10 所示的运行方法，查看本阶段的成果，如图 9.11 所示。

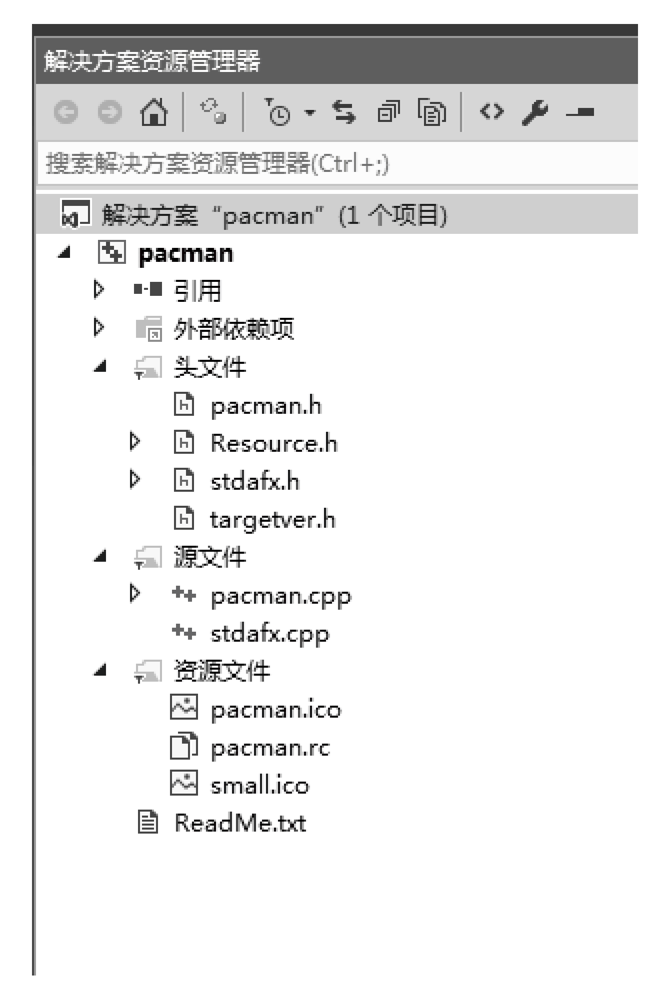


图 9.9　生成项目文件



图 9.10　运行方法



图 9.11　自动生成项目运行结果



9.4.2　wWinMain 函数

一个程序是如何实现的，首先要从程序的入口函数来观察。_tWinMain 是封装了 WinMain 的函数，此处不作详细讲解，其参数列表如下：

```
int APIENTRY wWinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine,
                      int nCmdShow)
```

HINSTANCE 代表的是程序的实例。第一个参数代表本程序的实例。第二个参数代表上一个程序的实例，是系统用来管理程序而使用的标识。第三个和第四个参数表示的是控制台参数的设置。这个函数就是窗口应用程序中的主函数，相当于控制台应用程序中的 Main 函数。

下面将介绍_tWinMain 函数中的参数。

首先可以看到的是两个宏。

```
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);
```

hPrevInstance 与 lpCmdLine 这两个参数几乎不会被用到，所以使用了一个 UNREFERENCED_PARAMETER 的宏，用来忽略来自编译器的“未使用过的参数”警报。

```
MSG msg;
HACCEL hAccelTable;
```

MSG 是一个结构体，它是 Windows 消息的记录形式，而下面的 HACCEL 是窗口中的热键表。

```
LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadString(hInstance, IDC_PACMAN, szWindowClass, MAX_LOADSTRING);
```

在工程项目中选中左侧的“资源视图”选项，可以看到这个项目包含的资源内容，如图 9.12 所示。在图 9.12 中，从上向下依次是热键表、对话框、图标、菜单和字符表，字符表中存储着一些字符数据。

资源视图
Accelerator
Dialog
Icon
Menu
String Table

图 9.12　资源视图

LoadString 函数用于将程序一开始定义的两个全局字符串变量 szTitle、szWindowClass 初始化为 String Table 中对应 ID 的字符串的值。需要重点讲解的是下面这个函数——MyRegisterClass(HINSTANCE hInstance)。

在本程序中所使用的窗体是已经定义好的一个窗口类，MyRegisterClass 函数完成了此项工作。在代码中可以找到这个函数，其内容如下：

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;
        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style = CS_HREDRAW | CS_VREDRAW;
```

```
        wcex.lpfnWndProc = WndProc;
        wcex.cbClsExtra = 0;
        wcex.cbWndExtra = 0;
        wcex.hInstance = hInstance;
        wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_PACMAN));
        wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName = MAKEINTRESOURCE(IDC_PACMAN);
        wcex.lpszClassName = szWindowClass;
        wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
        return RegisterClassEx(&wcex);
}
```

这个函数在主函数内调用，所接收的实参就是本程序的 HINSTANCE。WNDCLASSEX 是一个结构体，是所使用的窗口类，其参数在函数内初始化。

- ☑ cbSize：代表窗口类结构体所占大小。
- ☑ style：窗口的样式，CS_HREDRAW 、CS_VREDRAW 分别代表窗口在水平和竖直方向运动时，窗口的画面重绘。
- ☑ lpfnWndproc：指向窗口过程函数的指针。
- ☑ cbClsExtra：在窗口类后添加的字节数，经常置为 0。
- ☑ cbWndExtra：在窗口句柄后添加的字节数，同样经常置为 0。
- ☑ hInstance：在本应用程序的函数中，已经将其值传递到 MyRegisterClass 中。
- ☑ hIcon：程序图标。
- ☑ hCursor：鼠标的图标。
- ☑ hbrBackground：背景色。
- ☑ lpzsMenuName：菜单的名称。
- ☑ lpzsClassName：窗口类的名称。
- ☑ hIconSm：窗口的小图标。

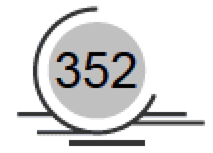
说明 在 Windows 窗口程序中，窗口、应用程序和菜单等都有自己的句柄。句柄是操作系统识别不同程序的标识。

这个结构体中设置了窗口的一般信息，最后使用函数 RegisterClassEx 注册了窗口类。这个函数是一个 Windows API（Applicaton Programming Interface）函数，是微软公司提供的 Windows 应用程序开发者的接口函数，具体实现则被封装起来。

可以观察到，在初始化窗口类参数时，使用了一些 API 函数，如下所示。

- ☑ LoadIcon：载入程序图标。
- ☑ MAKEINTRESOURCE：将资源菜单中相应 ID 的项作为资源。
- ☑ LoadCursor：载入鼠标图标。

如何更好地使用这些 API 不是本节的重点。相应地，理解 Windows 程序的工作原理更为重要。



注册完窗口类，主函数内又执行了这样一段代码：

```
if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
```

在代码中查找到名为 InitInstance 的函数，其内容如下：

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    hInst = hInstance;                    //将实例句柄存储在全局变  中
    hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
     CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
    if (!hWnd)
    {
      return FALSE;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    return TRUE;
}
```

HWND 是窗口句柄的缩写，是系统管理窗口的一个标识。之后使用了 Windows API 函数 CreateWindow 创建了一个窗口句柄，赋值给 hWnd 变量。CreateWindow 函数传入的实数的意义如下。

- ☑ szWindowClass：是本文件定义的一个 TCHAR 类型的数组，用来存储类名。
- ☑ szTitle：同样是全局变量，存储的是窗体的标题。
- ☑ WS_OVERLAPPEDWINDOW：该参数所对应的形参代表的是窗口的风格。
- ☑ 第 4、5 个参数表示窗口的左上角在屏幕中的位置，使用 CW_USDEFAULT 是使 x 坐标为默认值，这种情况下 y 坐标无须设置，可以置为 0。
- ☑ 第 6、7 个参数表示的是窗口横向和纵向大小，同样设为默认。
- ☑ 第 8 个参数代表这个窗口所属的父类窗口句柄，本窗口不使用，则设置为 NULL。
- ☑ 第 9 个参数代表使用的菜单句柄，本程序的菜单已在注册窗口类时通过资源加入进来，没有使用相应的句柄，则设置为 NULL。
- ☑ 第 10 个参数 hInstance 是本程序的标识，代表着创建的窗口属于哪个应用程序。
- ☑ 最后一个参数代表 Windows 参数，本例不使用。

如果创建失败返回 FALSE，则主函数会调用 return 语句结束程序。当创建成功后，主函数会调用 Windows API 函数 ShowWindow 来显示窗口，调用函数 UpdateWindow 来更新窗口的改动。

9.4.3　Windows 消息循环

运行程序之后，窗口一直显示的原因是在主函数中存在着一个消息循环。它不但维持着窗口程序的运行，还起着消息中转站的作用。这里所用到的函数都是 Windows API 函数。



```
while (GetMessage(&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

GetMessage 函数获取消息之后会通过 TranslateAccelerator 函数将消息与热键表对比，若不是热键发出的菜单指令，则会将这些消息传递给 TranslateMessage 函数并翻译出相应的形式，之后通过 DispatchMessage 函数传递给消息的处理者。

这些消息从何而来？大部分都是用户发出的，这些消息都放在消息队列中。消息会进入到队列，在这里等候被程序传递。

谁负责处理这些消息呢？前面用到的注册窗口类的窗口过程函数就是专门处理 Windows 消息的函数，每个窗口类都有自己的窗口过程函数，程序中主窗口的过程函数在如下代码中也可以被找到。

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {
        case WM_COMMAND:
            wmId = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            //分析菜单  择
            switch (wmId)
            {
            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        break;
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            //TODO: 在此添加任意绘图代码
            EndPaint(hWnd, &ps);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
```



```
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

返回值类型 LRESULT 是 long 指针，CALLBACK 是 stdcall 的宏，表示过程函数参数的压栈方式。各参数列表的意义如下。

☑ hWnd：是主窗口的句柄，表示这个过程函数是处理主窗口消息的。
☑ message：是无符号整型数据，它的数字代表着消息的类型。
☑ wParam 和 lParam：是与特定消息一起传递过来的参数的高位和低位，代表消息的具体信息。

在函数体内存在一个 Switch 语句，其作用是通过传递过来的消息类型作出反应。

以下是本程序中的各种情况代表的意义。

☑ WM_COMMAND：代表用户选择菜单项时所产生的信息，在其中使用参数的低位作为内容，如下所示。
  ➢ IDM_ABOUT：代表选择菜单项“关于”的消息，在本例中调用了 API 中的 DialogBox 函数创建了一个对话框。
  ➢ IDM_EXIT：代表选择菜单项“退出”的消息，在本例中调用了 API 中的 Destroy 函数销毁了 hWnd 所属的窗口，即主窗口。
☑ WM_PAINT：代表在窗口中绘图所产生的信息，本例中虽然使用了两个 API 函数，但它们并不对程序的表现有任何影响。
☑ WM_DESTROY：代表销毁窗口产生的信息，本例中使用了 PostQuitMessage 函数向消息队列中传递了销毁程序的消息。

在 Default 中，直接返回了 DefWindowProc 函数的结果。它也是 Windows API 函数，不难发现它的形式和窗口过程函数一样，实质上是默认的窗口过程函数。

消息队列、消息循环和回调函数之间的关系如图 9.13 所示。

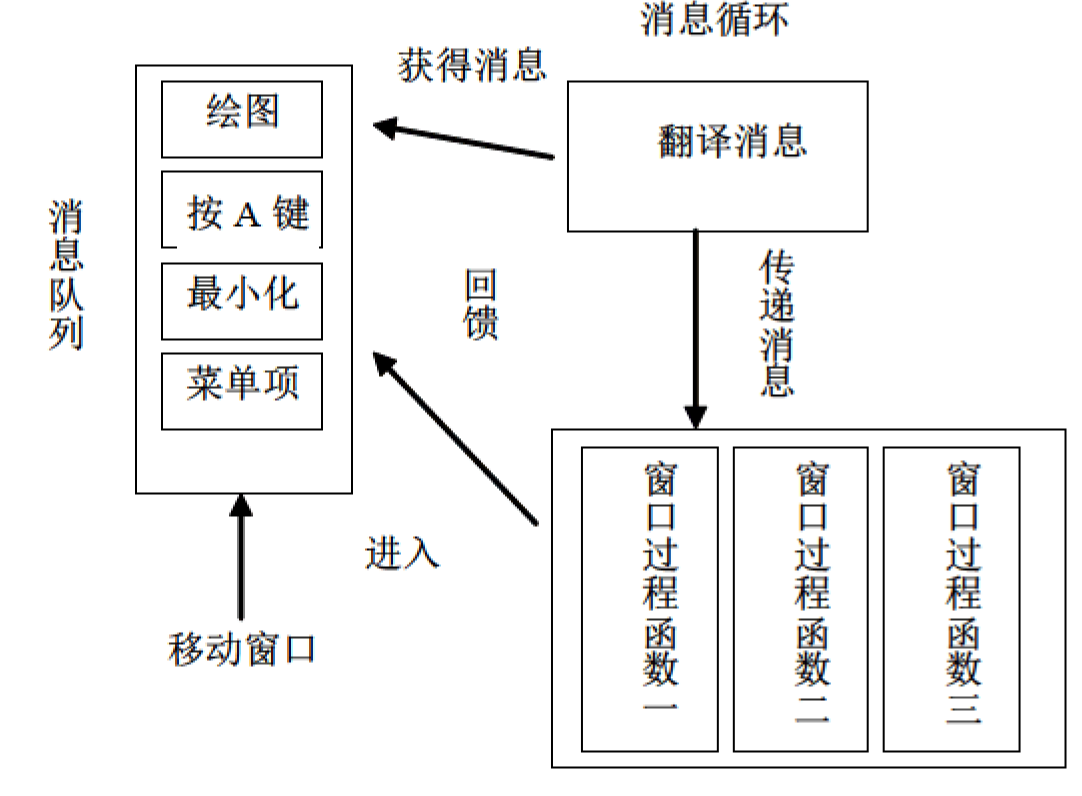


图 9.13　消息脉络

通过用户的操作产生的消息会进入队列，消息循环会从队列中拿走属于自己的消息。在消息循环中完成翻译的消息会被传递到相应的窗口过程函数中去，窗口函数负责在各种消息中作出反应，有时也要发回消息队列中。

在 Visual Studio 2017 编译器生成的 Windows 应用程序中，还有另外一个窗口过程函数，如下所示：

```
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
            UNREFERENCED_PARAMETER(lParam);
            switch (message)
            {
            case WM_INITDIALOG:
                  return (INT_PTR)TRUE;

            case WM_COMMAND:
                  if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                  {
                        EndDialog(hDlg, LOWORD(wParam));
                        return (INT_PTR)TRUE;
                  }
                  break;
            }
            return (INT_PTR)FALSE;
}
```

该函数是 About 对话框的窗口过程函数。在主窗口的窗口过程函数中处理 WM_COMMAND 消息时，使用的 DialogBox 函数的参数列表如下：

```
DialogBox(HINSTANCE hInst,lpTemplate MAKEINTRESOURCE(IDD_ABOUTBOX),
            HWND hWnd,lpDialogFun pFunc)
```

- ☑ HINSTANCE：应用程序的实例（句柄）。
- ☑ lpTemplate：对话框的模板指针。这里仍然使用了 MAKEINTRESOURCE 宏，将 ID 为 IDD_ABOUTBOX 的对话框资源作为模板使用。
- ☑ HWND：对话框所属父窗口的句柄。本程序的 About 对话框的父窗口即是已经创建完毕的主窗口框架。
- ☑ lpDialog：实质上是一个函数指针，其参数列表和返回值形式与窗口过程函数相同，代表对话框中使用的窗口过程函数。

在本项目中主要使用的是主窗口的消息过程函数，读者对 About 对话框的窗口过程函数稍做理解即可。

9.4.4 常用绘图 GDI

GDI（Grarph Device Interface）图形设备接口是 Windows API 中提供给开发者处理窗口程序的函数接口。

在使用 GDI 之前，需要让操作系统知道哪个窗口需要绘图。使用一个窗口句柄与设备联系起来，就可以完成这个任务。

HDC GetDC(HWND hWnd)是创建一个设备上下文的函数。它获得了绘图设备的句柄 HDC，可以在窗口上使用它开始绘图，例如：

```
HDC hdc = GetDC(HWND hWnd);
```

之后就可以使用变量 hdc 作为绘图函数中的标识，其真正意义是图形设备分配出来的资源标识。



下面介绍几个常用的绘图函数和结构体。

（1）点结构体 POINT：这个结构体包含两个属性，一个是横坐标 x，另一个是纵坐标 y，分别用来存储横、纵坐标的数据。

（2）若需要绘制一条直线，首先需要使窗口的光标先移动到起点，代码如下：

```
MoveToEx(HDC hdc,int x,int y,LPPOINT preP)
```

前 3 个参数分别代表设备上下文、挪动光标在窗口中的点的坐标，最后一个参数是前一个点指针的信息，一般置为空。

（3）将线“拉伸”到目标地点。代码如下：

```
LineTo(HDC hdc,int x,int y)
```

同样，参数依次代表设备上下文以及目标点的横、纵坐标。

（4）绘制矩形。矩形的结构体 RECT 的代码如下：

```
struct RECT
{
	int left;
	int top;
	int right;
	int bottom;
};
```

这个结构体由左、上、右、下 4 个值构成，分别代表了这个矩形 4 个方向的极值坐标，也可以理解为用左上角和右下角坐标的记录方式，如图 9.14 所示。

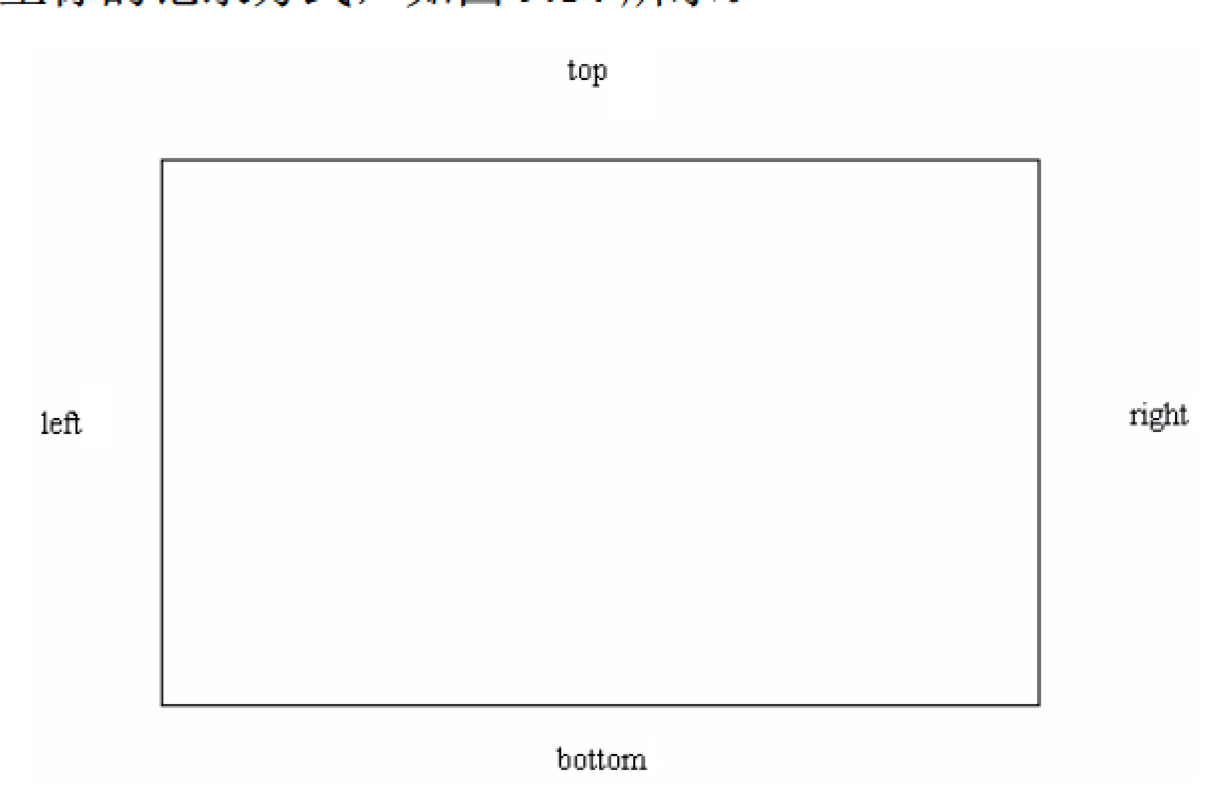


图 9.14　矩形结构体属性示意图

（5）绘制带颜色的实心矩形，代码如下：

```
FillRect(Hdc hdc,&RECT pRect,HBRUSH brush);
```

第一个和第二个参数分别代表图形设备上下文和轮廓矩形的指针。HBRUSH 是一个画刷资源，可以理解为画画的刷子。



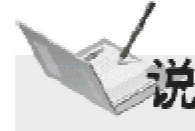
创建一个画刷的函数为：

```
CreateSolidBrush(COLORREF c)
```

COLORREF 是 GDI 中的颜色类型，使用 RGB 宏的编码可以设定它的颜色值。使用 CreateSolidBrush 并结合相应颜色可以创建一个实心的画刷。

绘制一个红色实心矩形，代码如下：

```
RECT rect;
...     //设置 rect 的位置信息
FillRect(hdc,&rect,CreateSolidBrush(RGB(255,0,0));
```



RGB 分别代表红色、绿色和蓝色在颜色中所占的比例，在这个宏中，取值的范围为 0～255。

（6）绘制椭圆，代码如下：

```
Ellipse(HDC hdc,int left,int top,int right,int bottom)
```

该函数绘制的圆形是一个以矩形为基准的内切圆，left、top、right 和 bottom 分别代表矩形的左、上、右、下点坐标，如图 9.15 所示。

（7）绘制圆弧，代码如下：

```
Arc(HDC hdc,int x1,int y1,int x2,int y2,
        int x3,int y3,int x4,int y4)
```

参数的意义如图 9.16 所示。

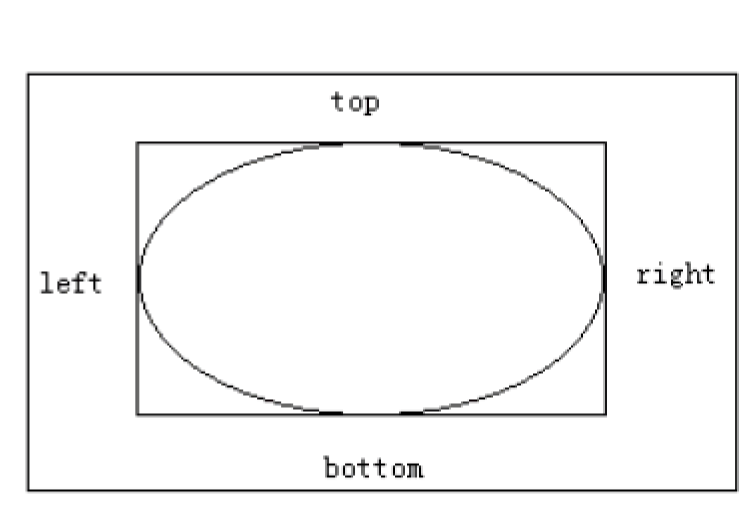


图 9.15　内切椭圆视图

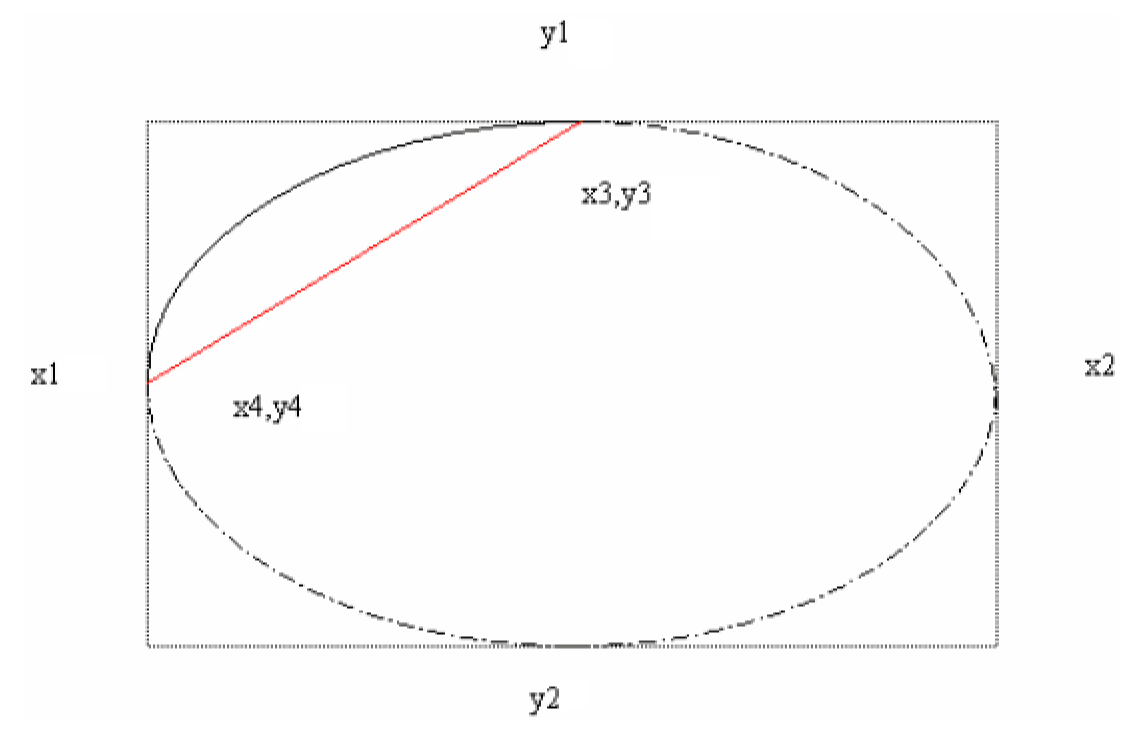


图 9.16　画弧函数参数示意图

Arc 画弧函数中需要填写的信息有以下 3 项：

☑　设备上下文。

☑　内切圆所在矩形的左、上、右、下信息。

☑　此段弧形对应的弦两端坐标，也可以理解为相应直线截取内切圆的交点。弧形依照逆时针方向绘制。



（8）画笔资源 HPEN。

GDI 绘图可以画出各种各样的线，这与程序中需用什么样的画笔有关。默认的画笔是黑色实线的。使用 CreatePen 函数可以创建一个画笔，代码如下：

```
CreatePen (int Style,int width, COLORREF c)
```

第一个参数是画笔的样式，0 代表实线；第二个参数代表线的宽度；第三个参数代表画笔的颜色。在程序中应用创建的画笔资源需要使用 SelectObject 函数。

```
HPEN SelectObject(HDC hdc,HGDIOBJ ob)
```

第二个参数代表的是绘图资源，如画刷、画笔和位图等，其返回值是 hdc 原本应用的资源。

以下是一个使用蓝色画笔绘制圆的例子。代码如下：

```
HPEN bluePen = CreatePen(0,1,RGB(0,0,255));
HPEN oldPen = (HPEN)SelectObject(hdc,bluePen);
RECT rect;
int x1 =5,x2=10,y1=5,y2=5;
Ellipse(hdc,x1,y1,x2,y2);
```

代码中的 oldPen 是绘图中应用过的画笔。当使用新画笔绘制完图形后，应当将画笔资源还原，并且释放创建的画笔资源，代码如下：

```
SelectObject(hdc,oldPen);
DeleteObject(bluePen);                        //  放画笔资源
```

DeleteObject 是一个释放画笔资源的函数，代码如下：

```
DeleteObject(HGDIOBJ ob)
```

以上是本项目中使用的 GDI 函数的内容，接下来将开始制作 PacMan。

9.4.5　碰撞检测的实现

在游戏中如何让程序知道物体在撞墙？可通过物体所在点的位置和墙体边缘位置进行检测。计算方法可以是中心坐标和朝向所对应的墙的位置与物体的宽度作比较，若是大于宽度，则没有碰上。这种碰撞检测方法虽然趋于真实，但是实现起来非常复杂。首先需要记录所有墙体边缘点的坐标（像素点），然后行走一步就判断一下朝向是否撞墙。其中还需要找到相应方向的墙壁，否则需要遍历所有墙壁边缘点的坐标。实现之后的情况可能还会存在一些不希望被观测到的结果，如转弯之后剩余半截身体卡在墙中等。

那么，如何设计吃豆子中的碰撞检测呢？地图的记录方式是关键。地图的记录数据量越少，计算的方法越简单。将整个地图分为若干个正方形的小方格，物体每到一个方格才会去做碰撞检测。地图记录的是这些小方格是否有墙体与豆子的信息，物体通过地图来判断是否撞到了墙壁。

本项目所设计的地图为长、高各 19 个方格，使用一个二维矩阵来记录，同时还应该记录地图中方格的大小。设计一个地图类，代码如下：

```
class Gmap
{
        protected:
                static int LD;                                    //  碍物尺寸
                bool mapData[MAPLENTH ][MAPLENTH ];               //  碍物    地图点
                frined class GObject;                             //将自  数据提供给友元物体类
};
```

其中，MAPLENTH 是数字 19 的宏。LD 是墙的尺寸。因为地图类所有对象的墙壁大小应该相同，所以应该将其设定为静态变量。

有了地图类，那么对于所有物体而言，它们使用的地图也应该是同一张。向 GObject 类添加地图类指针变量，代码如下：

```
protected:
        static GMap* pStage; //指向地图类的指   ，设置为   态，使所有子类对象   能够使用相同的地图
```

这样物体类就包含了地图的信息。在物体类中的碰撞检测依照地图类所记录方格信息进行判断。物体行进到一个方格中，判断它当前朝向的下一个格子是否有墙壁是当前碰撞的判断标准。在此之前需要判断的是物体是否在方格的中央，检测的标准如图 9.17 所示。

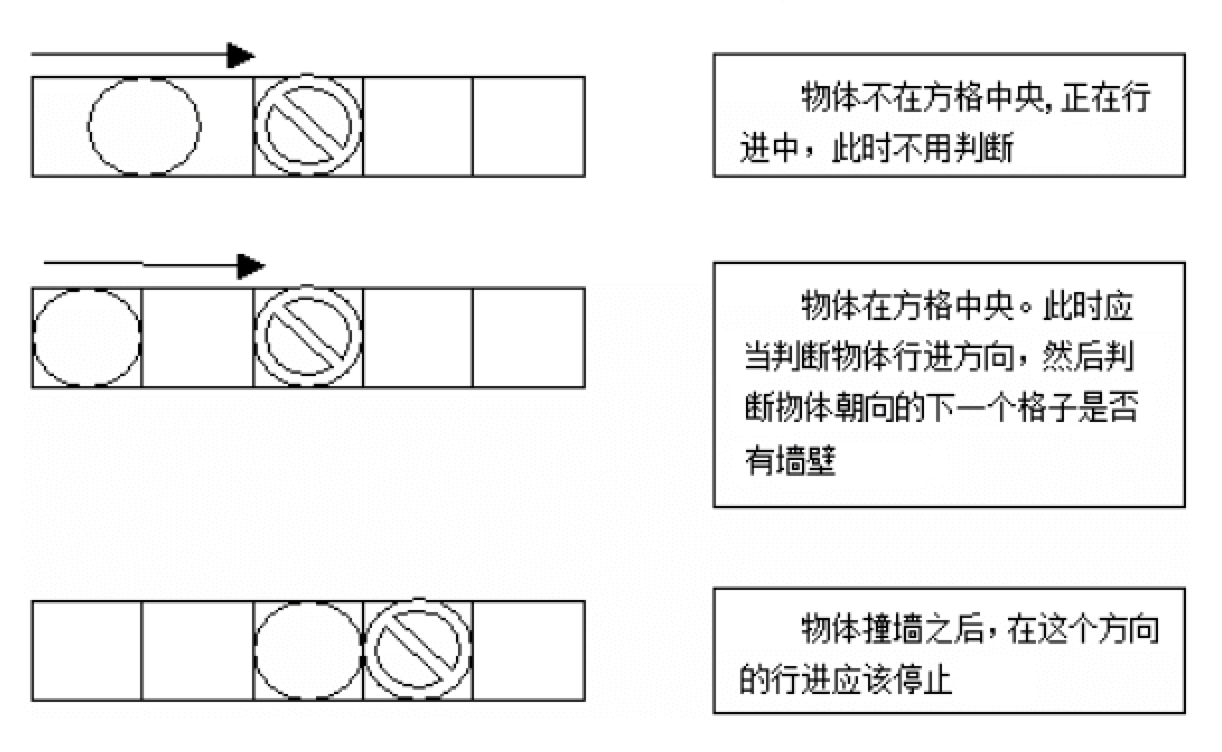
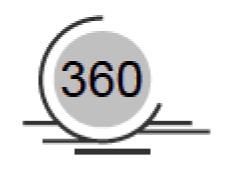


图 9.17　碰撞检测标准

现在需要记录物体在地图矩阵中的当前位置，并计算物体是否在方格中。在 Gobject 类中声明以下成员：

```
protected:
        bool Achive();                                //判断物体是否到      坐标位置
        int dRow;                                     //      横坐标（即所在矩   的行）
        int dArray;                                   //      纵坐标（即所在矩   的列）
        int speed;                                    //  度
        virtual void AchiveCtrl();                    //到        点后更新数据
```

将更新数据函数 AchiveCtrl 设定为虚函数。这个函数用于判断物体到达格子后，更新物体在矩阵中的行列坐标。玩家所操作的“大嘴”在到达方格后需要判断是否消除了豆子，这样等于在到达格子后添加了一种行为，这样很适合将此函数设置为虚函数供子类使用。



物体到达格子后除了向以前的方向前行，还可以转弯。在游戏中并不提倡随时都可以转弯，因为大多数情况一定是无效的指令（由于撞墙），所以在碰撞检测中也应该包含方向的更新和指令的有效性。“大嘴”和敌人都应该存在方向指令，这样才能够在地图上转弯。在物体类里可以使用一个前面定义的方向枚举 TWARDS 类型来存储这个指令。

```
TWARDS twCommand;                                         //指令缓存
```

在地图类 GMap 中，使用了 bool 型的二维矩阵存储地图上墙壁位置的信息。当值为 false 时，表示该位置有墙壁；当值为 true 时，说明该位置没有墙壁。

（1）依照碰撞检测的标准，首先应该更新物体所在的行、列的数据，代码如下：

```
bool GObject::Achive()
{
      int n =(point.x- pStage->LD/2)%pStage->LD;
      int k =(point.y- pStage->LD/2)%pStage->LD;
       bool l = (n==0&&k==0);
       return l;
}
void GObject::AchiveCtrl()
{
     if(Achive())
     {
          dArray = PtTransform(point.x);                   //更新列
          dRow = PtTransform(point.y);                     //更新行
     }
}
int GObject::PtTransform(int k)
{
     return (k -(pStage->LD)/2)/pStage->LD;
}
```

上述代码中，在类中将 PtTransform 函数的访问权限声明为 protected，其作用是将物体在屏幕上的坐标转换为行/列坐标。

以左上角第一个格子为基准进行坐标偏移与转换，如图 9.18 所示。

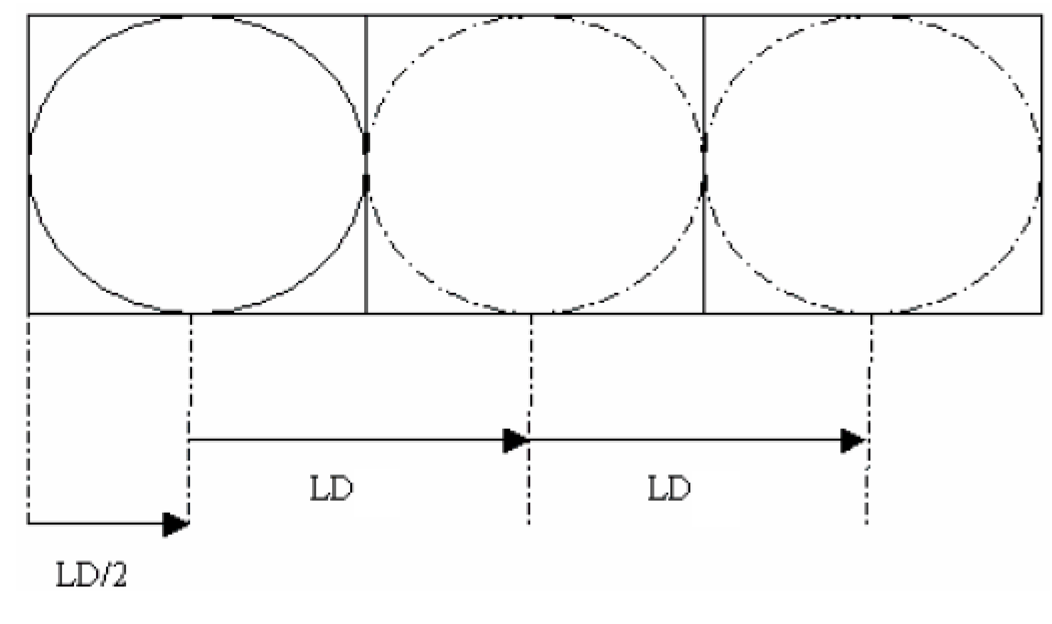


图 9.18　坐标偏移与转换

第一个格子的左上角坐标为 0。以列为例，当换算方格中心坐标与第一个方格的距离时，需要先减去第一个方格的左边界与中心坐标的距离，然后整除方格大小。

同样，在 Achive 函数中判断物体是否到达方格的判断条件是查看方格的中心是否和物体中心重合，如果重合，则应用矩阵坐标与窗口坐标的转换。

函数 AchiveCtrl 的作用是当物体到达方格中心时，更新物体的行列坐标。

（2）完成了坐标更新之后，还要查看当前指令的有效性。这个指令是在物体到达方格之前产生的，在物体到达方格时进行判断。在碰撞检测函数 Collision 中，应当先填写下列代码：

```
bool b = false;
    AchiveCtrl();//更新行、列的数据，若是“大嘴”，则会执行 PacMan   写的 AchiveCtrl 函数消   豆子
    //判断指令的有效性
    if(dArray<0||dRow<0||dArray>MAPLENTH||dRow>MAPLENTH)
    {
        b = true;
    }
else if(Achive())
    {
        switch(twCommand)                                       //判断行   的方向
        {
            case LEFT:
                if(dArray>0&&!pStage->mapData[dRow][dArray-1])  //判断下一个格子是否能够   行
                {
                    b = true;                                   //指令无效
                }
                break;
                //以下方向的判断原理相同
            case RIGHT:
                if(dArray<MAPLENTH-1&&!pStage->mapData[dRow][dArray+1])
                {
                    b = true;
                }
                break;
            case UP:
                if(dRow>0&&!pStage->mapData[dRow-1][dArray])
                {
                    b = true;
                }
                break;
            case DOWN:
                if(dRow<MAPLENTH-1&&!pStage->mapData[dRow+1][dArray])
                {
                    b = true;
                }
                break;
        }
        if(!b)
        {
```

```
            tw =twCommand;                                  //没撞墙，指令成功
        }
    }
```

以上是碰撞检测代码中检测方向指令有效性的片段。首先更新行列，若物体在屏幕外，则不可以改变指令。之后判断到达方格的物体指令方向的下一个格子是否有墙存在。若有墙存在，则指令无效；若没有撞墙，则指令成功，将方向替换成指令的方向。参数 b 除了可判断指令是否有效外，还会在设定人工智能时用到，稍后加以说明。

（3）之后物体应该朝着当前的方向继续前行。当下一个格子有墙壁出现时，物体不随速度 speed 改变位置。

```
switch(tw)                                                  //判断行　的方向
    {
        case LEFT:
            if(dArray>0&&!pStage->mapData[dRow][dArray-1])  //判断下一个格子是否能够　行
            {
                b= true;
                break;                                      //撞墙了
            }
            if(point.x<MIN)
            {
                point.x = MAX;
            }
            point.x -= speed;
            break;
            //以下方向的判断原理相同
        case RIGHT:
            if(dArray<MAPLENTH-1&&!pStage->mapData[dRow][dArray+1])
            {
                b= true;
                break;                                      //撞墙了
            }
            point.x += speed;
            if(point.x>MAX)
            {
                point.x = MIN;
            }
            break;
        case UP:
            if(dRow>0&&!pStage->mapData[dRow-1][dArray])
            {
                b= true;
                break;                                      //撞墙了
            }
            if(point.y<MIN)
            {
                point.y = MAX;
            }
```

```
                point.y -=speed;
                break;
            case DOWN:
                if(dRow<MAPLENTH-1&&!pStage->mapData[dRow+1][dArray])
                {
                    b= true;
                    break;                                         //撞墙了
                }
                point.y +=speed;
                if(point.y>MAX)
                {
                    point.y = MIN;
                }
                break;
        }
return b;
```

无须担心物体会因此卡在两格子中间而不能行动，因为在此之前已经更新过行列数据，只有在物体到达格子中央时才更新。MAX 和 MIN 分别代表超出地图边界一个方格的位置。当物体超过地图边界到达地图外时，则会从另一边出现。



9.5　制作 PacMan

9.5.1　PacMan 程序框架初步分析

制作软件之前，都需要对软件需求作较为全面的分析。游戏的效果如图 9.1～图 9.3 所示。

图中弧形的物体形象地体现了“大嘴”的角色，其他彩色的类似章鱼的物体则是敌人，过道中的小圆圈是豆子。使用键盘的方向键对游戏进行操作，“大嘴”路过并可以“吃掉”豆子，但触碰到敌人则会失败。

以面向对象的设计方法来思考，吃豆子中的所有会移动的物体具有很多共同的特性，列举如下：

☑　会移动。

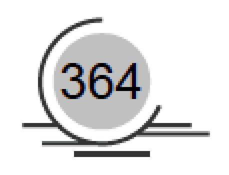
☑　移动分为上、下、左、右 4 个方向。

☑　碰到墙和障碍物会停止。

☑　物体拥有自己的坐标，这些物体所处的层面是相同的：拥有绘图效果，但各不相同；各个物体都有自己的移动准则，例如，“大嘴”是玩家控制的，而敌人则是依照设定好的人工智能行动。

这些物体完全不同的方面有：“大嘴”能吃豆子，敌人不能；敌人能够抓住“大嘴”，“大嘴”不能够抓住任何物体。

依照以上列出的条件，可以设计一个物体类，本项目中取名为 GObject，它是游戏中可移动物体的父类。将共性放入父类中，则该类应该具有的属性有：在平面地图上所处位置、自身的朝向和碰撞检测。现在用代码描述这个父类（即声明成员）并建立一个方向枚举：



```
enum TWARDS {
    UP,     //上
    DOWN,   //下
    LEFT,   //左
    RIGHT, //右
    OVER,   //游戏结束
};
class GObject
{
public:
    GObject(int Row, int Array)
    {
        //帧数
        m_nFrame = 1;
        //当前关卡
        pStage = NULL;
        //行
        this->m_nRow = Row;
        //数组
        this->m_nArray = Array;
        //中心位置
        this->m_ptCenter.y = m_nRow * pStage->LD + pStage->LD / 2;
        this->m_ptCenter.x = m_nArray * pStage->LD + pStage->LD / 2;

        this->m_nX = m_ptCenter.x;
        this->m_nY = m_ptCenter.y;
    }

    //设置位置
    void SetPosition(int Row, int Array);
    //画空白
    void DrawBlank(HDC &hdc);
    void virtual Draw(HDC &hdc) = 0;      //绘制对象
    void virtual action() = 0;            //数据变更的表现

    int GetRow();
    int GetArray();

    static GMap *pStage;          //指向地图类的指  ，设置为  态，使所有类对象  能够使用相同的地图

protected:
    int m_nX;
    int m_nY;
    //指令缓存
    TWARDS m_cmd;
    //中心坐标
    POINT m_ptCenter;
    //     横坐标
```

```
    int m_nRow;
    //      纵坐标
    int m_nArray;
    //   度
    int m_nSpeed;
    //朝向
    TWARDS m_dir;
    //帧数
    int m_nFrame;
    //判断物体是否到        坐标位置
    bool Achive();
    //      碰撞检测，将物体摆放到合理的位置
    bool Collision();
    //将实   坐标   换为      坐标
    int PtTransform(int k);
    //到        点后更新数据
    virtual void AchiveCtrl();
};
```

9.5.2　建立游戏循环

游戏中需要快速地显示画面和更新游戏状态，因此需要一个一直运行的循环来驱动画面和状态的更新。在创建 Win32 程序时，已经自动生成了一个消息循环，主要用来处理 Windows 系统消息。修改这个消息循环，使它既可以处理 Windows 消息，又可以作为游戏循环使用。

打开 pacman.cpp 文件，找到：

```
//主消息循环
while(GetMessage(&msg, nullptr, 0, 0)) {
  if(!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
}
```

上面的代码是 Visual Studio 2017 自动生成的窗口消息循环。这个循环中使用的 GetMessage 函数只有在有消息时才会返回，无消息时阻塞。把上述代码替换成如下代码：

```
//主消息循环
bool bRunning = true;
while(bRunning) {
  //获取消息
  if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
    if(msg.message == WM_QUIT) {
      break;
    }
    TranslateMessage(&msg);
```



```
        DispatchMessage(&msg);
    }
}
```

9.6　使用 GDI 绘图



后面的开发过程，需要绘制各种图形，这些复杂的图形都是由简单的图形组合而成。因此先实践一下这些“简单”图形绘制函数。本程序中使用 Windows 系统提供的 GDI 库完成绘图工作。GDI 库包含一组函数，可以完成简单的绘制工作。这些函数可以在工程中直接使用。

9.6.1　画点

点是最简单的图形，理论上可以使用点组合成任意图形。画点的函数是 SetPixel，该函数可以在指定坐标的位置画一个指定颜色的点。该函数使用方法如下。

在 pacman.cpp 文件中找到：

```
//获取消息
if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
    if(msg.message == WM_QUIT) {
        break;
    }
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

这一段是消息循环代码，在此段代码下面增加如下代码：

```
//画点测试
{
    HDC hdc = ::GetDC(g_hwnd);                                    //获得设备句柄
    SetPixel(hdc, rand() % WLENTH , rand() % WHIGHT,              //在  机的位置画一个  机  色的点
            RGB(rand() % 256, rand() % 256, rand() % 256));
    ::ReleaseDC(g_hwnd, hdc);                                     //  放设备
}
```

上述代码实现生成随机坐标位置和随机颜色值，并向窗口输出点的功能。

9.6.2　画矩形

要画矩形，可以画 4 条直线，首尾相接即可，读者感兴趣可以自己实验。这里使用另外两个 API 画矩形。

☑　Rectangle：画的是一个矩形框。

☑ FillRect：填充矩形。

下面尝试画一个空心的矩形，一个填充的矩形。将上面画线测试部分的代码换成：

```
//画矩形测试
{
    HDC hdc = ::GetDC(g_hwnd);
    {
        //创建画笔
        HPEN pen = CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
        //   择画笔
        HPEN oldPen = (HPEN)SelectObject(hdc, pen);
        //画矩形（空心）
        Rectangle(hdc, 100, 200, 300, 500);
        //恢复画笔
        SelectObject(hdc, oldPen);
        DeleteObject(pen);
    }

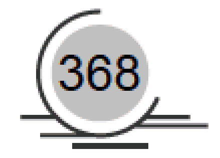
    {
        //创建画笔
        HBRUSH bBrush = CreateSolidBrush(RGB(0, 0, 255));
        //填充矩形
        RECT rect;
        rect.left = 50;
        rect.top = 270;
        rect.right = 150;
        rect.bottom = 370;
        FillRect(hdc, &rect, bBrush);
        DeleteObject(bBrush);
    }

    ::ReleaseDC(g_hwnd, hdc);
    //暂停 1 毫秒，不然画得太快，看不清
    Sleep(1);
}
```

9.6.3 画圆

程序中的“豆子”和“敌军”对象的眼睛部分用到了画圆函数，画圆函数的原型如下：

```
BOOL Ellipse(
    _In_ HDC hdc,
    _In_ int nLeftRect,
    _In_ int nTopRect,
    _In_ int nRightRect,
    _In_ int nBottomRect
);
```



其中，后 4 个参数为左上、右下两点的坐标。如果这两个点组成一个正方形，则画出来的就是圆，如果是长方形，则画出来的是椭圆。下面进行测试，将上面画矩形测试部分换成画圆测试：

```
//画圆测试
{
    HDC hdc = ::GetDC(g_hwnd);
    //画圆：后　4 个数字，构成一个正方形
    Ellipse(hdc, 200, 150, 300, 250);
    //画椭圆
    Ellipse(hdc, 200, 270, 340, 370);
    //画椭圆
    Ellipse(hdc, 100, 100, 200, 150);
    ::ReleaseDC(g_hwnd, hdc);
}
```

9.6.4　画弧型

本程序中的玩家对象是一个不断吞咬的“大嘴”形象，某些时刻，“大嘴”上面是一段弧线，而不是整个圆。可以认为弧型是椭圆形上面的一小段，画弧型的函数原型如下：

```
BOOL Arc(
    _In_ HDC hdc,
    _In_ int nLeftRect,             //左上点坐标 x
    _In_ int nTopRect,              //左上点坐标 y
    _In_ int nRightRect,            //右下点坐标 x
    _In_ int nBottomRect,           //右下点坐标 y
    _In_ int nXStartArc,            //　始点坐标 x
    _In_ int nYStartArc,            //　始点坐标 y
    _In_ int nXEndArc,              //结束点坐标 x
    _In_ int nYEndArc               //结束点坐标 y
);
```

该函数的 nLeftRect、nTopRect、nRightRect 和 nBottomRect 参数确定了一个矩形，进而确定一个椭圆，弧形为该椭圆上面截取的一段。后面 4 个参数确定了该段的起始点和结束点。

下面用该函数输出两个弧形。将上面画圆形测试部分换成：

```
//画弧形测试
{
    HDC hdc = ::GetDC(g_hwnd);
    Arc(hdc, 100, 100, 200, 300          //矩形左上点，右下点
      , 150, 200                         //　点
      , 100, 200                         //终点（与　点　时　　接）
     );

    Arc(hdc, 0, 0, 100, 100
```

```
    , 50, 100
    , 50, 0
    );
  ::ReleaseDC(g_hwnd, hdc);
}
```

9.6.5　画玩家

尝试使用上面的知识，在游戏窗口中画出自己控制的玩家对象。玩家对象的形象是一个大嘴机器人，如图 9.19 所示。要想画出这个对象，只需要在窗口上交替画出 3 个形状即可。这 3 个形状是由前面讲述的圆、直线和弧形组合而成的。

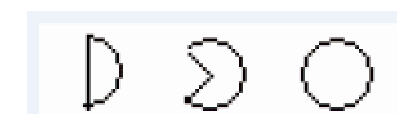


图 9.19　游戏中的形状

下面画出一个放大版的玩家对象。绘画过程共分 5 帧，模拟闭嘴的形状、张嘴的形状、完全张开嘴的形状。将上面画弧型测试部分换成如下代码部分：

```
//综合应用，画一个大嘴对象
{
  static DWORD dwTime = GetTickCount();
  //当　离上次绘图的时　大于 50 毫秒时，才　行本次绘制
  if(GetTickCount() - dwTime >= 50) {
    dwTime = GetTickCount();
  }
  else {
    continue;
  }
  /* 模拟当前的帧
    本对象一共 5 帧，每一帧画不同的图形
  */
  static int iFrame = 0;
  ++iFrame;
  if(iFrame >= 5) {
    iFrame = 0;
  }

  //代表对象的中心位置
  int x = 300, y = 300;

  //对象的半径
  int r = 100;

  //dc 对象句柄
  HDC hdc = ::GetDC(g_hwnd);
```

```
    std::shared_ptr<HDC__> dc(::GetDC(g_hwnd), [](HDC hdc) {
        ::ReleaseDC(g_hwnd, hdc);
    });
    //获取窗口客户区大小
    RECT rc;
    GetClientRect(g_hwnd, &rc);

    //创建画刷
    std::shared_ptr<HBRUSH__> br(
        ::CreateSolidBrush(RGB(255, 255, 255)),
      [](HBRUSH hbr) {
        ::DeleteObject(hbr);
    });

    //画背景（清　上一帧所画内容）
    FillRect(dc.get(), &rc, br.get());

#define PI (3.1415926f)                                             //定义圆周率的值
    switch(iFrame) {
        case 0: {
            Ellipse(dc.get(), x - r, y - r, x + r, y + r);          //画一个圆
            MoveToEx(dc.get(), x - r, y, NULL);                     //画一个横线
            LineTo(dc.get(), x, y);
            break;
        }
        case 1: {
            //画嘴（两条线与纵　偏离 PI/4）
            int x0, y0;                                             //左上角的点
            int x1, y1;                                             //左下角的点
            x0 = x - static_cast<int>(r * sin(PI * 0.75f));
            y0 = y + static_cast<int>(r * cos(PI * 0.75f));

            x1 = x + static_cast<int>(r * sin(PI * 1.25f));
            y1 = y - static_cast<int>(r * cos(PI * 1.25f));

            SetPixel(dc.get(), x0, y0, RGB(255, 0, 0));
            SetPixel(dc.get(), x1, y1, RGB(0, 255, 0));
            SetPixel(dc.get(), x, y, RGB(0, 0, 0));
            Arc(dc.get(), x - r, y - r, x + r, y + r                //画一个半圆和一条竖线
                , x1, y1
                , x0, y0);

            MoveToEx(dc.get(), x0, y0, NULL);                       //画竖线
            LineTo(dc.get(), x, y);

            MoveToEx(dc.get(), x1, y1, NULL);
            LineTo(dc.get(), x, y);
            break;
```

```
}
case 2: {
  Arc(dc.get(), x - r, y - r, x + r, y + r                    //画一个半圆和一条竖线
    , x, y + r
    , x, y - r
    );
  //画竖线
  MoveToEx(dc.get(), x, y - r, NULL);                         //从圆弧上  的点开始
  LineTo(dc.get(), x, y + r);                                 //到圆弧下  的点结束
  break;
}
case 3: {
  //画嘴（两条线与纵  偏离 PI/4）
  int x0, y0;                                                 //左上角的点
  int x1, y1;                                                 //左下角的点
  x0 = x - static_cast<int>(r * sin(PI * 0.75f));
  y0 = y + static_cast<int>(r * cos(PI * 0.75f));

  x1 = x + static_cast<int>(r * sin(PI * 1.25f));
  y1 = y - static_cast<int>(r * cos(PI * 1.25f));

  SetPixel(dc.get(), x0, y0, RGB(255, 0, 0));
  SetPixel(dc.get(), x1, y1, RGB(0, 255, 0));
  SetPixel(dc.get(), x, y, RGB(0, 0, 0));
  //画一个半圆和一条竖线
  Arc(dc.get(), x - r, y - r, x + r, y + r
    , x1, y1
    , x0, y0);

  //画竖线
  MoveToEx(dc.get(), x0, y0, NULL);
  LineTo(dc.get(), x, y);

  MoveToEx(dc.get(), x1, y1, NULL);
  LineTo(dc.get(), x, y);
  break;

}
case 4: {
  //画一个圆
  Ellipse(dc.get(), x - r, y - r, x + r, y + r);
  //画一条横线
  MoveToEx(dc.get(), x - r, y, NULL);
  LineTo(dc.get(), x, y);
  break;
}
default:
```

```
            break;
        }
    }
```

我们把整个过程分成 3 部分，如图 9.20～图 9.22 所示，然后反复绘制这一过程，最后看起来的动画效果就是玩家控制的大嘴对象。

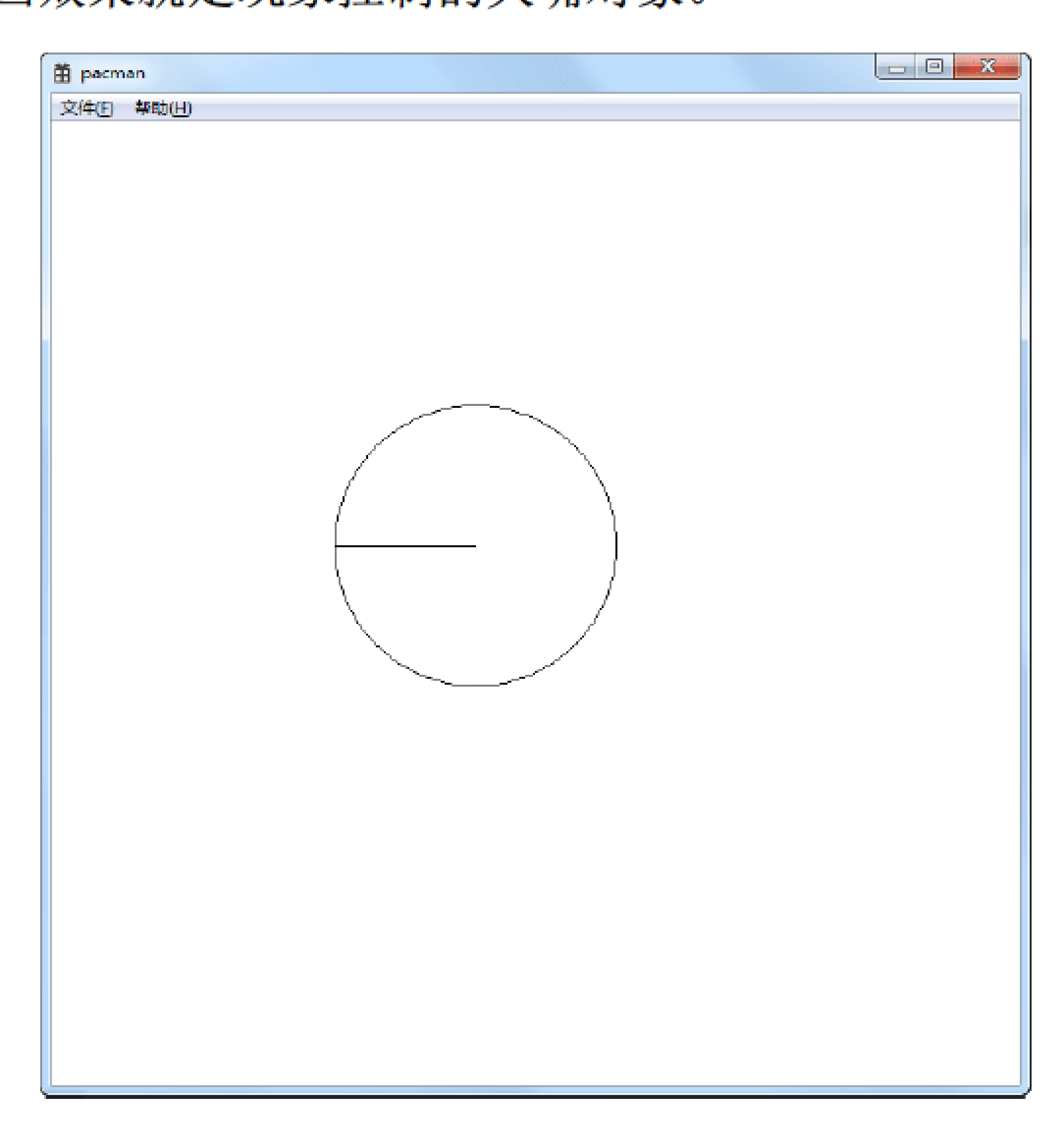


图 9.20　绘制玩家形象 1

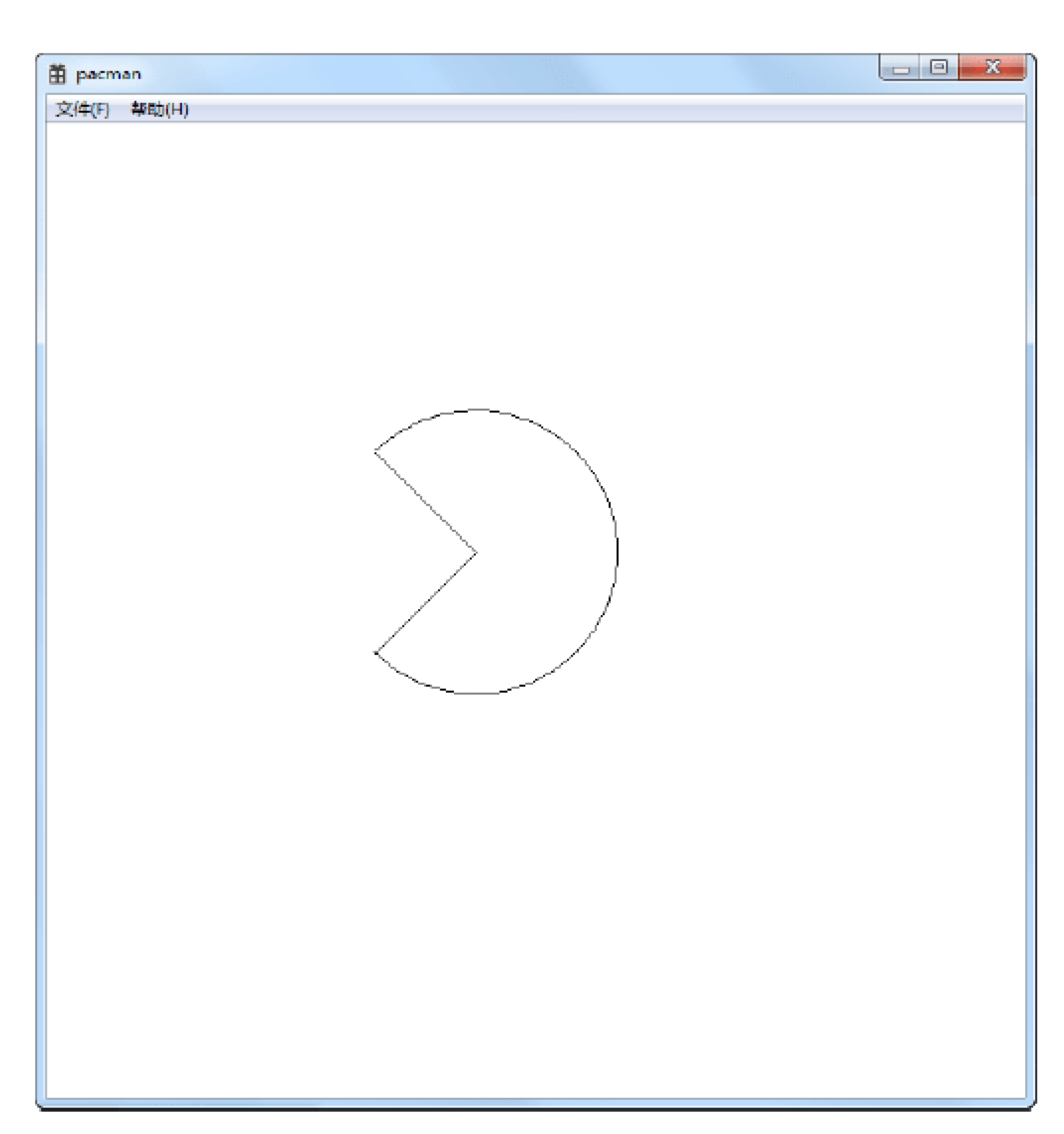


图 9.21　绘制玩家形象 2

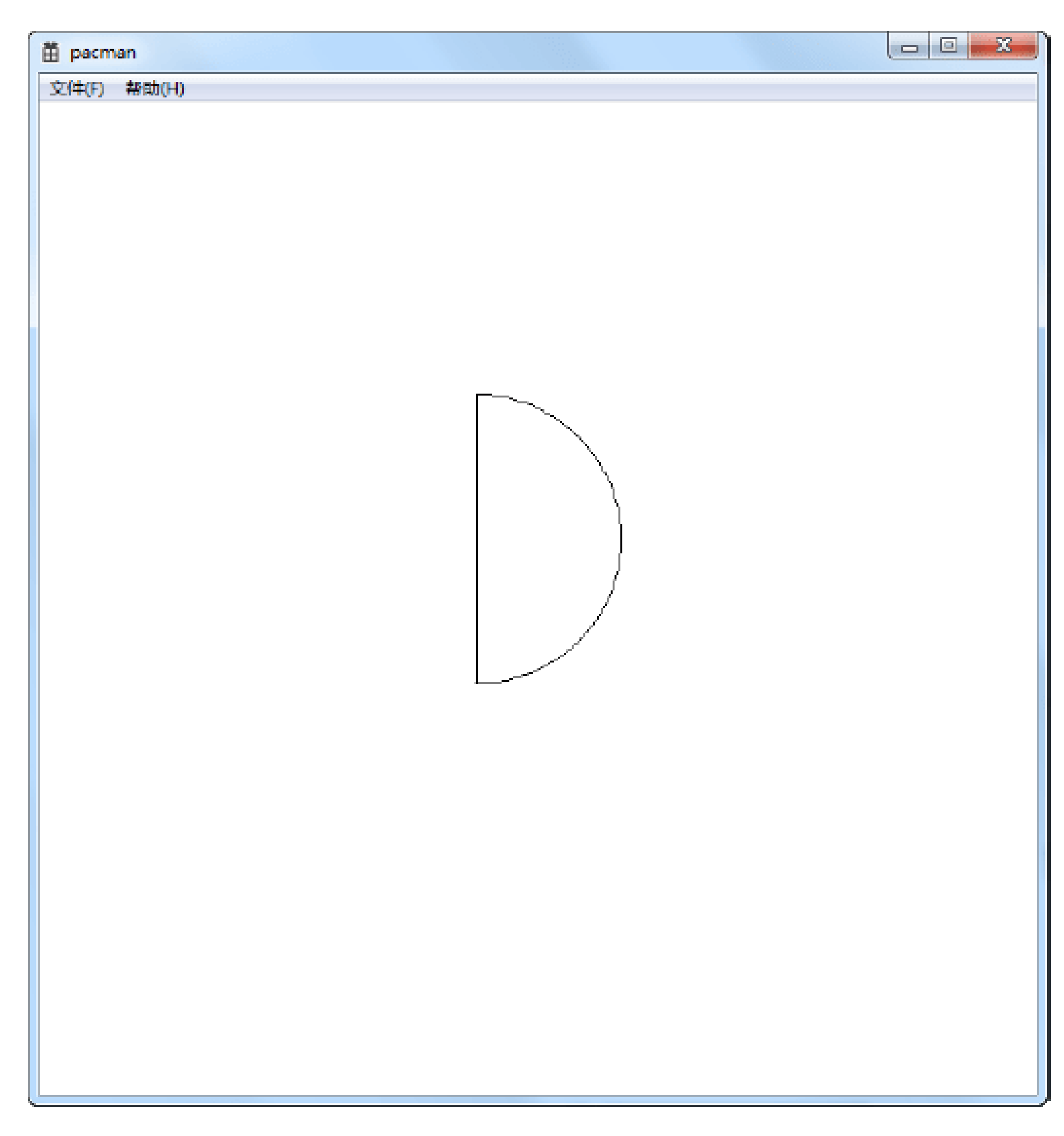


图 9.22　绘制玩家形象 3


视频讲解

9.7 地图及关卡制作

本游戏场景设定在一个小迷宫中，其中有墙、门和豆子。玩家和敌军在地图中移动，遇到墙则需要变换方向，玩家遇到豆子就吃掉豆子，留下一块空白区域。这些功能全都集中在地图类中。游戏中一共设定了 3 个类似的地图，为游戏的 3 个关卡。下面设计地图类，并设定三关的数据。

9.7.1 地图类设计

本程序中共设计了三关。每关的地图不同但游戏逻辑相同，因此需要在地图类中增加一个父类，并扩展出 3 个子类，存放三关数据。在工程中增加 GMap 类。

增加地图类的声明。用于声明障碍物的尺寸、豆子的半径、地图的初始函数、地图中障碍物的数组和地图中豆子的数组等。打开 GMap.h 文件，输入：

```
#pragma once

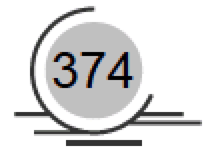
#include <list>

#define MAPLENTH 19                          //    地图大小
#define P_ROW 10                             //我方的位置坐标
#define P_ARRAY 9                            //我方的位置坐标
#define E_ROW 8                              //敌方的位置坐标
#define E_ARRAY 9                            //敌方的位置坐标

using std::list;

//抽象类 GMap
class GMap
{

protected:
    static int LD;                           //   碍物尺寸
    static int PD;                           //豆子的半径
    void InitOP();                           //敌我双方出现位置没有豆子出现
    bool mapData[MAPLENTH][MAPLENTH];        //   碍物      地图点
    bool peaMapData[MAPLENTH][MAPLENTH];     //豆子      地图点
    COLORREF color;                          //地图中墙的    色
public:
    void  DrawMap(HDC &hdc);                 //绘制地图
    void  DrawPeas(HDC &hdc);                //绘制豆子
    virtual ~GMap();
    GMap()
    {
```



```
    }
    friend class GObject;                    //允许物体类使用直线的  点和终点的信息做碰撞检测
    friend class PacMan;                     //允许“大嘴”访  豆子地图
};
```

9.7.2　第一关地图的设计

在 GMap.h 文件中接着输入第一关地图的声明。第一关地图的类名是 Stage_1，该类为地图类 GMap 的子类，类中声明了静态数组，该数组中包含地图的初始数据。具体代码如下：

```
//第一关
class Stage_1 : public GMap
{
private:
    bool static initData[MAPLENTH][MAPLENTH];    //地图数据
public:
    Stage_1();
};
```

9.7.3　第二关地图的设计

在 GMap.h 文件中接着输入第二关地图的声明。第二关地图的类名是 Stage_2，该类为地图类 GMap 的子类，类中声明了静态数组，该数组中包含地图的初始数据。

```
//第二关
class Stage_2 : public GMap
{
private:
    bool static initData[MAPLENTH][MAPLENTH];    //地图数据
public:
    Stage_2();
};
```

9.7.4　第三关地图的设计

在 GMap.h 文件中接着输入第三关地图的声明。第三关地图的类名是 Stage_3，该类为地图类 GMap 的子类，类中声明了静态数组，该数组中包含地图的初始数据。

```
//第三关
class Stage_3 : public GMap
{
private:
```

```
    bool static initData[MAPLENTH][MAPLENTH];            //地图数据
public:
    Stage_3();
};
```

9.7.5 地图类的实现

（1）下面是地图及关卡类的具体实现。打开 GMap.cpp 文件，输入以下代码。其中，LD 为墙的宽度，PD 为豆子的直径，这两个是静态成员，需要在这里进行初始化。

```
#include "stdafx.h"
#include "GMap.h"

int GMap::LD = 36;                                       //墙的宽度
int GMap::PD = 3;                                        //豆子的直径
```

（2）地图中的 peaMapData 数组为豆子的数据，玩家刚出现的位置不需要有豆子，因此需要把数组中这个位置的元素设为 false，代表此处没有豆子。在 GMap.cpp 文件最下方输入以下代码：

```
//敌我双方出现位置没有豆子出现
void GMap::InitOP()
{
    peaMapData[E_ROW][E_ARRAY] = false;                  //敌方位置没有豆子
    peaMapData[P_ROW][P_ARRAY] = false;                  //玩家位置没有豆子
}
```

（3）类的成员变量 mapData 存储了墙体的数据。遍历这个数组，当发现该处是墙壁时，在此位置会看到一个矩形模拟墙体。在 GMap.cpp 文件最下方接着输入绘制地图函数：

```
void GMap::DrawMap(HDC &memDC)
{
    HBRUSH hBrush = CreateSolidBrush(color);
    for(int i = 0; i < MAPLENTH; i++) {
        for(int j = 0; j < MAPLENTH; j++) {
            //绘制墙壁
            if(!mapData[i][j]) {
                RECT rect;
                rect.left = j * LD;
                rect.top = i * LD;
                rect.right = (j + 1) * LD;
                rect.bottom = (i + 1) * LD;
                FillRect(memDC, &rect, hBrush);          //填充矩形区域，模拟墙体
            }
        }
    }
```

```
    DeleteObject(hBrush);                                    //删　画刷对象
}
```

（4）成员变量 peaMapData 存储的是豆子数据。遍历该数组，如果发现该处元素为真，则调用画圆的函数画豆子。在 GMap.cpp 文件最下方接着输入绘制“豆子”的函数代码：

```
void GMap::DrawPeas(HDC &hdc)                                //画豆子函数
{
    for(int i = 0; i < MAPLENTH; i++) {                      //　历整个数组
        for(int j = 0; j < MAPLENTH; j++) {
            if(peaMapData[i][j]) {                           //如果该处有豆子

                Ellipse(hdc, (LD / 2 - PD) + j * LD,         //画圆：模拟豆子
                        (LD / 2 - PD) + i * LD,
                        (LD / 2 + PD) + j * LD,
                        (LD / 2 + PD) + i * LD);
            }
        }
    }
```

9.7.6　游戏　藏后　的实现

该游戏虽然看起来简单，实际上由于豆子很多，敌军速度很快，要想轻松过关还是有一定难度的。为了帮助玩家快速通关，设计一个游戏的后门：在按下 B 键时，直接通过当前关。在 GMap.cpp 文件最下方输入后门代码如下：

```
    //如果按下 B　，直接　关
    if(GetAsyncKeyState('B') & 0x8000) {
        MessageBoxA(NULL, "无意中您发现了秘笈", "", MB_OK);
        for(int i = 0; i < MAPLENTH; i++) {
            for(int j = 0; j < MAPLENTH; j++) {
                peaMapData[i][j] = false;
            }
        }
    }
}
```

接着输入析构函数：

```
GMap::~GMap()
{

}
```

游戏隐藏后门的运行效果如图 9.23 所示。

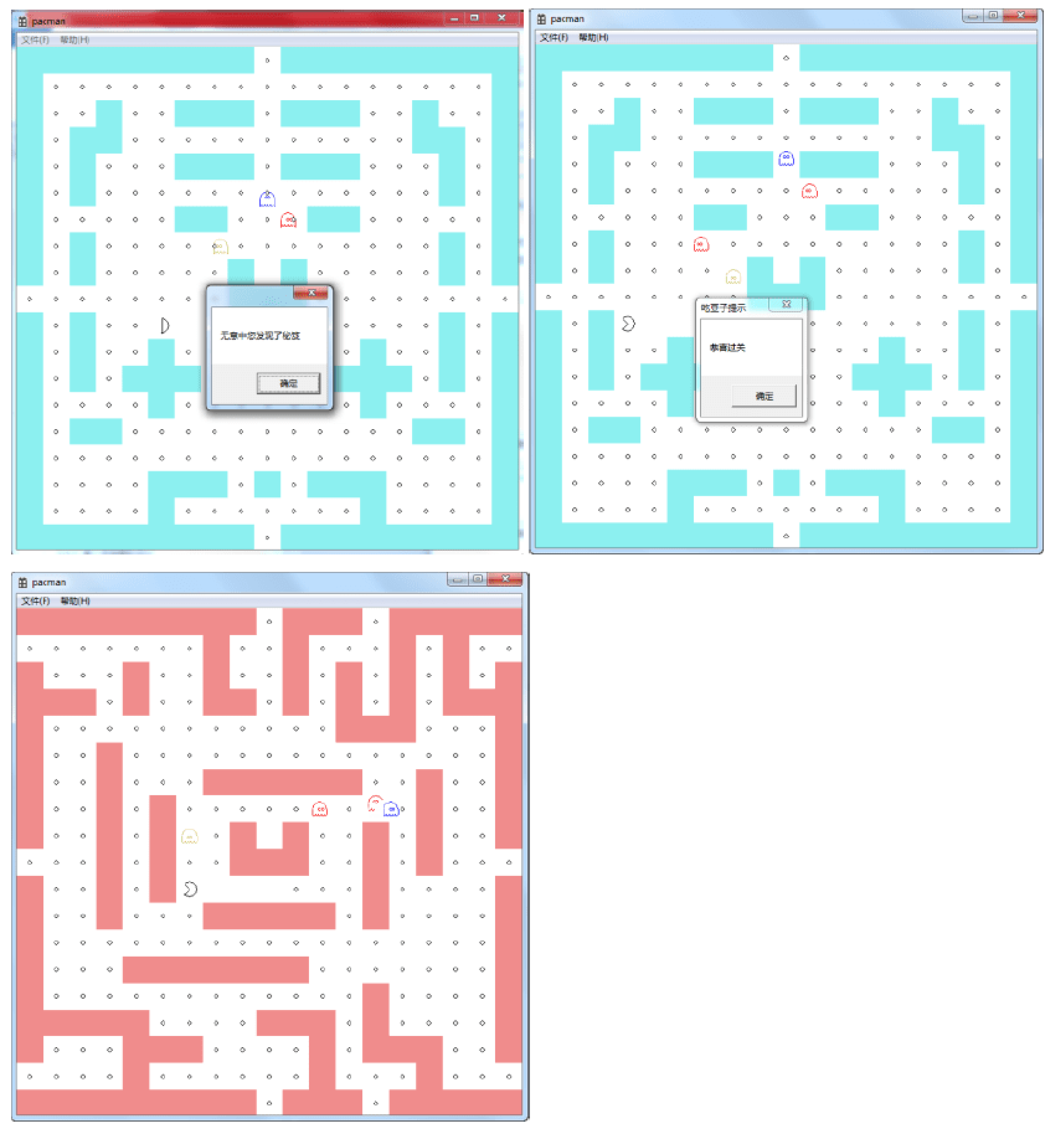


图 9.23　游戏后门的运行效果

9.7.7　第一关地图的实现

在 GMap.cpp 文件最下方输入第一关地图相关函数及数据，代码中定义了 A 为真，B 为假，其中真的位置代表该处有豆子，假的位置代表该处是墙：

```
//Stage_1 成员定义
#define A true                                                    //true：表示豆子
#define B false                                                   //false：表示墙壁
bool Stage_1::initData[MAPLENTH][MAPLENTH] = {
   B, B, B, B, B, B, B, B, B, A, B, B, B, B, B, B, B, B, B,     //0
   B, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, B,     //1
   B, A, A, B, A, A, B, B, B, A, B, B, B, A, A, B, A, A, B,     //2
   B, A, B, B, A, A, A, A, A, A, A, A, A, A, A, B, B, A, B,     //3
   B, A, B, A, A, A, B, B, B, A, B, B, B, A, A, A, B, A, B,     //4
   B, A, B, A, A, A, A, A, A, A, A, A, A, A, A, A, B, A, B,     //5
```



```
    B, A, A, A, A, A, B, B, A, A, A, B, B, A, A, A, A, A, B,    //6
    B, A, B, A, A, A, A, A, A, A, A, A, A, A, A, A, B, A, B,    //7
    B, A, B, A, A, A, A, A, B, A, B, A, A, A, A, A, B, A, B,    //8
    A, A, A, A, A, A, A, A, B, B, B, A, A, A, A, A, A, A, A,    //9
    B, A, B, A, A, A, A, A, A, A, A, A, A, A, A, A, B, A, B,    //10
    B, A, B, A, A, B, A, A, A, A, A, A, A, B, A, A, B, A, B,    //11
    B, A, B, A, B, B, B, A, A, A, A, A, B, B, B, A, B, A, B,    //12
    B, A, A, A, A, B, A, A, A, A, A, A, A, B, A, A, A, A, B,    //13
    B, A, B, B, A, A, A, A, A, A, A, A, A, A, A, B, B, A, B,    //14
    B, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, B,    //15
    B, A, A, A, A, B, B, B, A, B, A, B, B, B, A, A, A, A, B,    //16
    B, A, A, A, A, B, A, A, A, A, A, A, A, B, A, A, A, A, B,    //17
    B, B, B, B, B, B, B, B, B, A, B, B, B, B, B, B, B, B, B,    //18
};
#undef A
#undef B
Stage_1::Stage_1()
{
    color = RGB(140, 240, 240);                                 //墙的　色
    for(int i = 0; i < MAPLENTH; i++) {
        for(int j = 0; j < MAPLENTH; j++) {
            this->mapData[i][j] = this->initData[i][j];
            this->peaMapData[i][j] = this->initData[i][j];
        }
    }
    //敌我双方出现位置没有豆子出现
    this->InitOP();
}
```

第一关地图效果如图 9.24 所示。

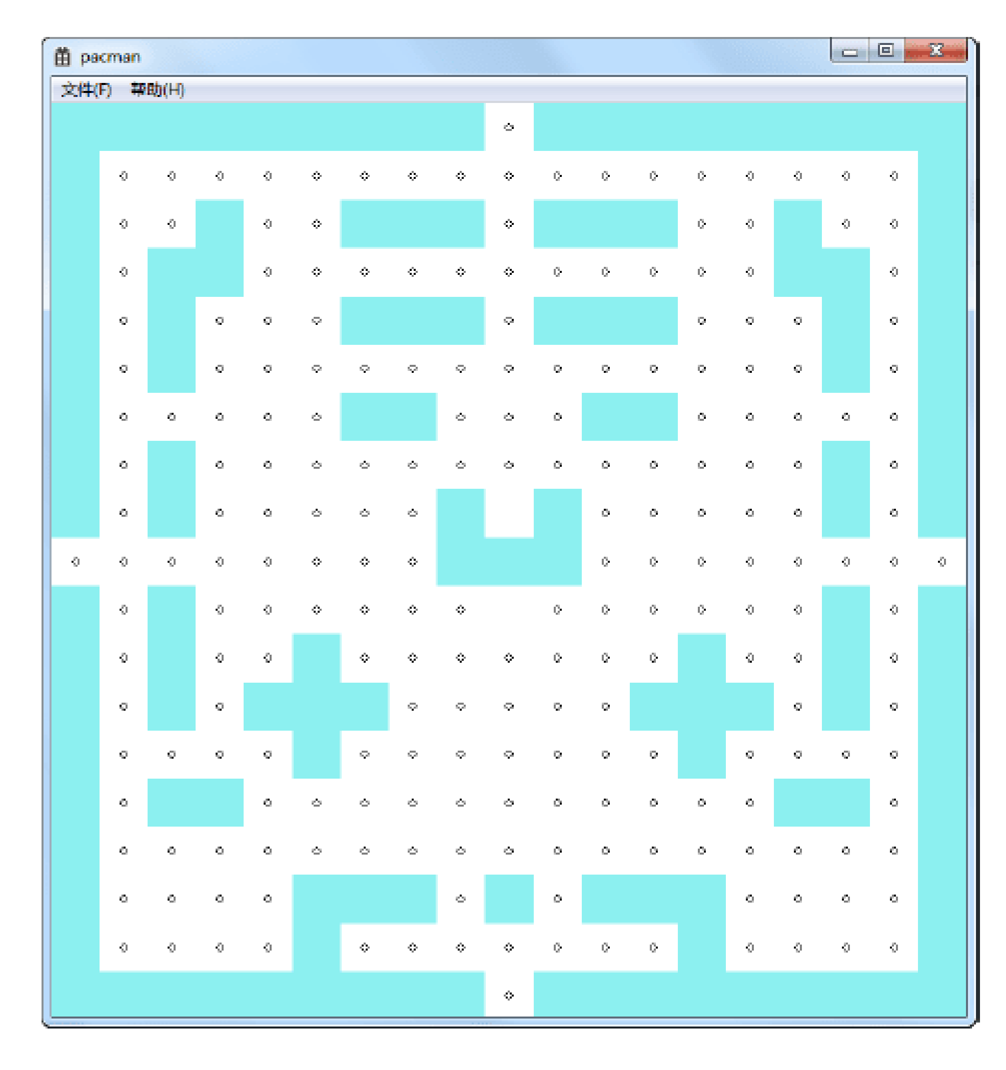


图 9.24　第一关地图

9.7.8 第二关地图的实现

在 GMap.cpp 文件最下方接着输入第二关地图相关函数及数据，代码中定义了 A 为真，B 为假，其中真的位置代表该处有豆子，假的位置代表该处是墙：

```
//Stage_2 成员定义
#define A true
#define B false
bool Stage_2::initData[MAPLENTH][MAPLENTH] = {
   B, B, B, B, B, B, B, B, B, A, B, B, B, A, B, B, B, B, B, //0
   A, A, A, A, A, A, A, B, A, A, B, A, A, A, B, A, B, A, A, //1
   B, A, A, A, B, A, A, B, A, A, B, A, B, A, B, A, B, A, B, //2
   B, B, B, A, B, A, A, B, B, A, B, A, B, A, B, A, B, B, B, //3
   B, A, A, A, A, A, A, A, A, A, A, A, B, B, B, A, A, A, B, //4
   B, A, A, B, A, A, A, A, A, A, A, A, A, A, A, A, A, A, B, //5
   B, A, A, B, A, A, A, B, B, B, B, B, B, A, A, B, A, A, B, //6
   B, A, A, B, A, B, A, A, A, A, A, A, A, A, A, B, A, A, B, //7
   B, A, A, B, A, B, A, A, B, A, B, A, A, B, A, B, A, A, B, //8
   A, A, A, B, A, B, A, A, B, B, B, A, A, B, A, B, A, A, A, //9
   B, A, A, B, A, B, A, A, A, A, A, A, A, B, A, A, A, A, B, //10
   B, A, A, B, A, A, A, B, B, B, B, B, A, B, A, A, A, A, B, //11
   B, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, B, //12
   B, A, A, A, B, B, B, B, B, B, B, A, A, A, A, A, A, A, B, //13
   B, A, A, A, A, A, A, A, A, A, A, A, A, B, A, A, A, A, B, //14
   B, B, B, B, B, A, A, A, A, B, B, B, A, B, A, A, A, A, B, //15
   B, A, A, A, B, B, B, A, A, A, A, B, A, B, B, B, A, A, B, //16
   A, A, A, A, B, A, A, A, A, A, A, B, A, A, A, B, A, A, A, //17
   B, B, B, B, B, B, B, B, B, A, B, B, B, A, B, B, B, B, B, //18
};
#undef A
#undef B
Stage_2::Stage_2()
{
   color = RGB(240, 140, 140);                              //墙的  色
   for(int i = 0; i < MAPLENTH; i++) {
      for(int j = 0; j < MAPLENTH; j++) {
         this->mapData[i][j] = this->initData[i][j];
         this->peaMapData[i][j] = this->initData[i][j];
      }
   }
   //敌我双方出现位置没有豆子出现
   this->InitOP();
}
```

第二关地图效果如图 9.25 所示。



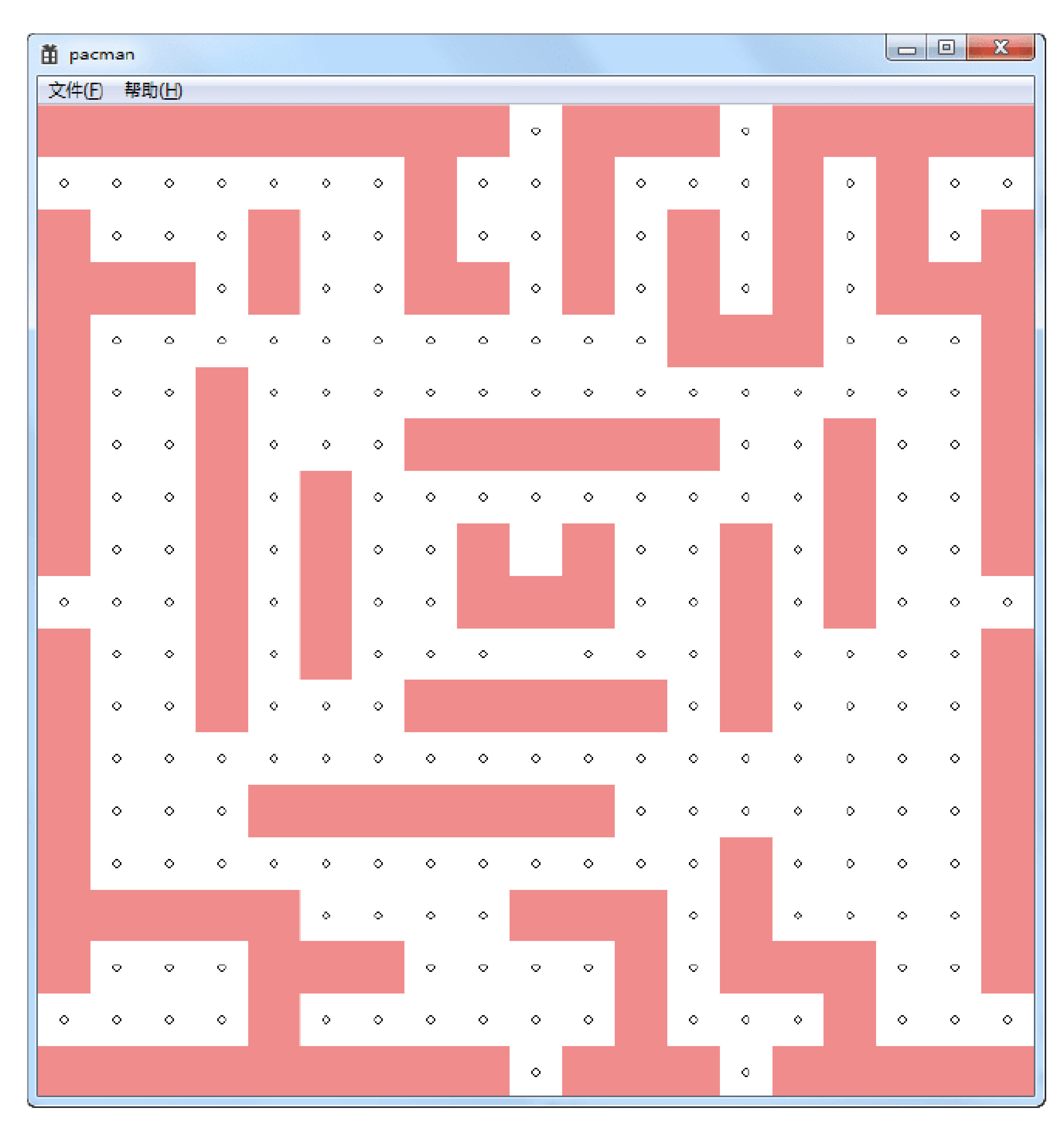


图 9.25　第二关地图

9.7.9　第三关地图的实现

在 GMap.cpp 文件最下方接着输入第三关地图相关函数及数据，代码中定义了 A 为真，B 为假，其中真的位置代表该处有豆子，假的位置代表该处是墙：

```
//Stage_3 成员定义
#define A true
#define B false
bool Stage_3::initData[MAPLENTH][MAPLENTH] = {
   B, B, B, B, B, B, B, B, B, A, B, B, B, B, B, B, B, B, B, //0
   A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, //1
   B, A, A, B, A, A, B, B, B, B, B, B, B, A, A, A, B, A, B, //2
   B, A, B, B, A, A, A, A, A, A, A, A, B, A, A, A, B, A, B, //3
   B, A, B, A, A, A, B, B, B, B, B, B, B, A, A, A, B, A, B, //4
   B, A, B, A, B, B, B, A, A, A, A, A, B, B, B, A, B, A, B, //5
   B, A, A, A, B, A, B, A, A, A, A, A, A, A, A, A, B, A, B, //6
   B, A, B, A, B, A, A, A, A, A, A, A, A, B, A, A, B, A, B, //7
   B, A, B, A, B, B, A, A, B, A, B, A, A, B, A, A, B, A, B, //8
   B, A, A, A, A, B, A, A, B, B, B, A, A, B, A, A, B, A, B, //9
   B, A, B, A, A, B, A, A, A, A, A, A, A, B, A, A, A, A, B, //10
   B, A, B, A, A, B, A, A, A, A, A, A, B, B, B, A, B, A, B, //11
   B, A, B, A, A, B, A, B, B, B, B, B, B, A, B, A, B, A, B, //12
```

```
    B, A, B, A, A, B, A, A, A, A, A, A, A, A, B, A, B, A, B, //13
    B, A, B, B, A, B, B, B, B, B, B, A, B, A, B, A, B, A, B, //14
    B, A, A, A, A, B, A, A, A, A, A, A, B, A, B, A, B, A, B, //15
    B, B, B, B, B, B, A, A, B, B, B, A, B, A, B, A, B, A, B, //16
    A, A, A, A, A, A, A, A, B, A, A, A, A, A, B, A, A, A, A, //17
    B, B, B, B, B, B, B, B, B, A, B, B, B, B, B, B, B, B, B, //18
};
#undef A
#undef B
Stage_3::Stage_3()
{
    color = RGB(100, 44, 100);                                  //墙的  色
    for(int i = 0; i < MAPLENTH; i++) {
        for(int j = 0; j < MAPLENTH; j++) {
            this->mapData[i][j] = this->initData[i][j];
            this->peaMapData[i][j] = this->initData[i][j];
        }
    }
    //敌我双方出现位置没有豆子出现
    this->InitOP();
}
```

第三关地图效果如图 9.26 所示。

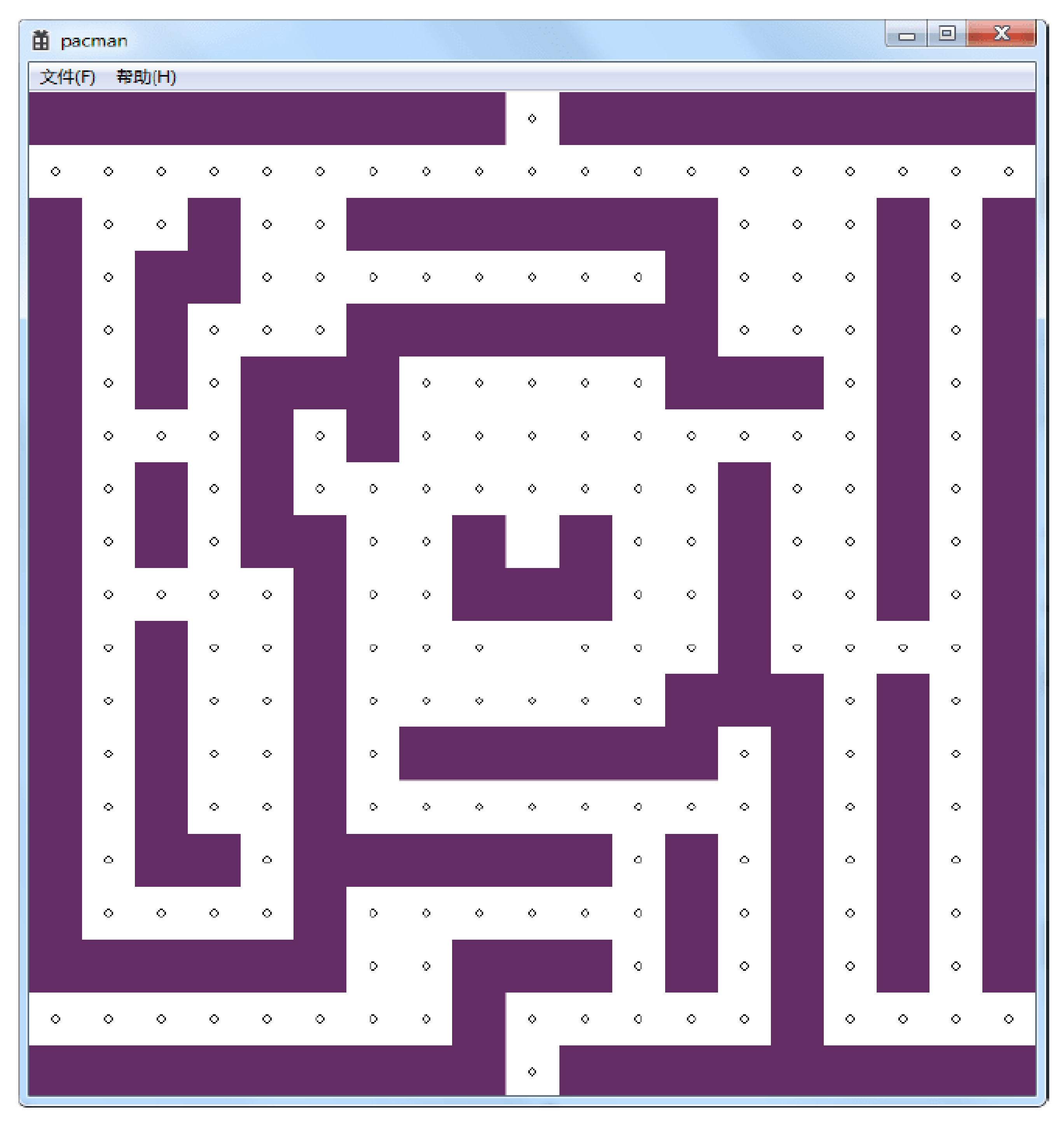
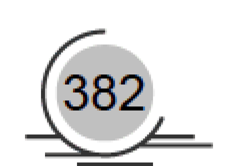


图 9.26　第三关地图



9.7.10　使用地图

打开 pacman.cpp 文件，增加包含头文件 GMap.h 的代码（在#include "pacman.h"一行下面插入#include "GMap.h"）。

找到“HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_PACMAN));”一行，把此行以下（不包括本行）直到函数尾部的内容全部删除，增加以下代码：

```
//当前的关卡
int s_n = 0; //[0, 1, 2]
//地图
GMap *MapArray[STAGE_COUNT] = { new Stage_1(), new Stage_2(), new Stage_3() };

MSG msg;

//主消息循环
bool bRunning = true;
while(bRunning && s_n < STAGE_COUNT) {
  //获取消息
  if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
    if(msg.message == WM_QUIT) {                    //WM_QUIT 消息，  出循环
      break;
    }
    TranslateMessage(&msg);                         //翻译消息
    DispatchMessage(&msg);                          //分发消息
  }
  HDC hdc = ::GetDC(g_hwnd);
  {
    MapArray[s_n]->DrawPeas(hdc);                   //画豆子
    MapArray[s_n]->DrawMap(hdc);                    //画地图
  }
  ::ReleaseDC(g_hwnd, hdc);                         //  放设备资源
}

return (int) msg.wParam;
```

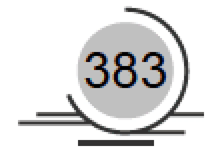
上述代码中定义了一个地图对象数组 MapArray，并在每次消息循环中调用地图的绘制方法，分别绘制地图。

9.8　游戏可移动对象设计与实现


视频讲解

9.8.1　可移动对象的设计

游戏中的可移动对象包括敌军和玩家对象，同属于游戏中可移动、可绘制的对象，因此可以抽象



出一个共同的父类 GObject 来存放两种对象的相同逻辑和数据。在工程中增加 GObject 类。

打开 GObject.h 文件，删除原来的内容，输入：

```
#include "stdafx.h"
#include <time.h>

#include "GMap.h"
#define PLAYERSPEED 6                        //玩家　度
#define ENERMYSPEED 4                        //敌人　度
#define LEGCOUNTS 5                          //敌人腿的数
#define DISTANCE 10                          //图形范围
#define BLUE_ALERT 8                         //蓝色警戒范围
#define D_OFFSET   2                         //绘图误差
#define RD (DISTANCE + D_OFFSET)             //绘图范围 12

enum TWARDS {                                //方向枚举
    UP,                                      //上
    DOWN,                                    //下
    LEFT,                                    //左
    RIGHT,                                   //右
    OVER,                                    //游戏结束
};

class GObject                                //物体类：大嘴和敌人的父类
{
public:
    GObject(int Row, int Array)
    {
        m_nFrame = 1;                        //帧数
        pStage = NULL;                       //当前关卡
        this->m_nRow = Row;                  //行
        this->m_nArray = Array;              //数组
        //中心位置
        this->m_ptCenter.y = m_nRow * pStage->LD + pStage->LD / 2;
        this->m_ptCenter.x = m_nArray * pStage->LD + pStage->LD / 2;

        this->m_nX = m_ptCenter.x;
        this->m_nY = m_ptCenter.y;
    }

    void SetPosition(int Row, int Array);    //设置位置
    void DrawBlank(HDC &hdc);                //画空白
    void virtual Draw(HDC &hdc) = 0;         //绘制对象
    void virtual action() = 0;               //数据变更的表现

    int GetRow();
    int GetArray();

    static GMap *pStage; //指向地图类的指　，设置为　态，使所有子类对象　能够使用相同的地图
```

```
protected:
    int m_nX;
    int m_nY;
    TWARDS m_cmd;                              //指令缓存
    POINT m_ptCenter;                          //中心坐标
    int m_nRow;                                //      横坐标
    int m_nArray;                              //      纵坐标
    int m_nSpeed;                              //   度
    TWARDS m_dir;                              //朝向
    int m_nFrame;                              //帧数
    bool Achive();                             //判断物体是否到      坐标位置
    bool Collision();                          //      碰撞检测，将物体摆放到合理的位置
    int PtTransform(int k);                    //将实   坐标   换为      坐标
    virtual void AchiveCtrl();                 //到         点后更新数据
};
```

上面代码即声明 GObject 类的代码，其中定义了共同属性，如坐标、速度和指令等；还定义了共同函数，如画空白和设置位置等。注意其中的含有 virtual 关键字的函数声明，下面的子类将会根据各自不同的逻辑覆盖该函数。

9.8.2　玩家对象的设计

在 GObject.h 文件中接着输入玩家对象的声明。玩家对象 PacMan 为 GObject 的子类。该类扩展了父类的功能，实现了 Draw 函数和 action 函数，其中 Draw 函数负责绘制自己，action 函数负责本类的行为。本类的构造函数中，设置了本类的初始速度为 PLAYERSPEED，设置了朝向为 LEFT。具体代码如下：

```
class PacMan : public GObject                  //玩家对象
{
protected:
    virtual void AchiveCtrl();                 //   写虚函数

public:
    POINT GetPos();
    bool IsOver();                             //游戏是否结束
    bool IsWin();                              //玩家是否赢得游戏
    void Draw(HDC &hdc);                       //负责绘制自己
    void SetTwCommand(TWARDS command);         //设置玩家下一步指令
    PacMan(int x, int y) : GObject(x, y)       //构   函数，产生新对象时调用
    {
        this->m_nSpeed = PLAYERSPEED;          //设置玩家   度
        m_cmd = m_dir = LEFT;
    }
    void action();                             //玩家的动作函数
    void SetOver();                            //设置游戏结束函数
};
```

9.8.3 可移动对象的实现

下面是 GObject 对象的实现代码。主要功能包括以下几种。

（1）位置调整函数 AchiveCtrl：将物体摆放到合理的位置。

（2）DrawBlank：绘制空白区域。

（3）Collision：碰撞检测。

打开 GObject.cpp 文件，输入 GObject 对象实现代码：

```
#include "stdafx.h"
#include "GObject.h"

//GOject 成员定义
GMap *GObject::pStage = NULL;

int GObject::GetRow()                                       //  回行
{
    return m_nRow;
}

int GObject::GetArray()                                     //  回数组   地址
{
    return m_nArray;
}

int GObject::PtTransform(int k)                             //坐标   换函数
{
    return (k - (pStage->LD) / 2) / pStage->LD;

}
//判断物体是否到        坐标位置
bool GObject::Achive()
{
    int n = (m_ptCenter.x - pStage->LD / 2) % pStage->LD;   //计算 x 坐标的余数
    int k = (m_ptCenter.y - pStage->LD / 2) % pStage->LD;   //计算 y 坐标的余数
    bool l = (n == 0 && k == 0);                            //如果两个余数   为 0，说明到   中心位置
    return l;
}
//到        坐标后更新数据
void GObject::AchiveCtrl()
{
    if(Achive()) {                                          //如果到        坐标
        m_nArray = PtTransform(m_ptCenter.x);               //更新列
        m_nRow = PtTransform(m_ptCenter.y);                 //更新行
    }
}

void GObject::DrawBlank(HDC &hdc)
```

```
{
    //申请资源，并交给智能指  处理
    HBRUSH hbr = ::CreateSolidBrush(RGB(255, 255, 255));    //创建画刷，绘制矩形函数要求使用
    std::shared_ptr<HBRUSH> phbr(&hbr, [](auto hbr) {       //把资源交给智能指  处理，自动  放
        DeleteObject(*hbr);                                 //离开 DrawBlank 函数时，会自动调用  放资源
    });
    RECT rect;
    rect.top = m_nY - RD;
    rect.left = m_nX - RD;
    rect.right = m_nX + RD;
    rect.bottom = m_nY + RD;
    FillRect(hdc, &rect, *phbr);                            //绘制矩形
}

//设置中心位置
void GObject::SetPosition(int Row, int Array)
{
    m_nRow = Row;
    m_nArray = Array;
    this->m_ptCenter.y = m_nRow * pStage->LD + pStage->LD / 2;
    this->m_ptCenter.x = m_nArray * pStage->LD + pStage->LD / 2;
}

//碰撞检测
bool GObject::Collision()
{
    bool b = false;

    //更新行和列的数据，若是大嘴，则会执行 PacMan  写的 AchiveCtrl 函数消  豆子
    AchiveCtrl();
    //判断指令的有效性
    if(m_nArray < 0 || m_nRow < 0 || m_nArray > MAPLENTH - 1
      || m_nRow > MAPLENTH - 1) {
        b = true;
    }
    else if(Achive()) {
        switch(m_cmd) { //判断行  的方向
            case LEFT:  //如果朝向为左
                //判断下一个格子是否能够  行
                if(m_nArray > 0 &&
                  !pStage->mapData[m_nRow][m_nArray - 1]) {
                   b = true;                                //撞墙了
                }
                break;
            //以下方向的判断原理相同
            case RIGHT: //如果朝向为右
                if(m_nArray < MAPLENTH - 1 &&
                  !pStage->mapData[m_nRow][m_nArray + 1]) {
                   b = true;                                //撞墙了
```

```
            }
            break;
        case UP:            //如果朝向为上
            if(m_nRow > 0 &&
              !pStage->mapData[m_nRow - 1][m_nArray]) {
               b = true;                                        //撞墙了
            }
            break;
        case DOWN:          //如果朝向为下
            if(m_nRow < MAPLENTH - 1 &&
              !pStage->mapData[m_nRow + 1][m_nArray]) {
               b = true;                                        //撞墙了
            }
            break;
    }
    if(!b) {
        m_dir = m_cmd;                                          //没撞墙，指令成功
    }
}
//依照真实的方向位移
m_nX = m_ptCenter.x;
m_nY = m_ptCenter.y;
int MAX = pStage->LD * MAPLENTH + pStage->LD / 2;
int MIN = pStage->LD / 2;
switch(m_dir) {  //判断行　的方向
    case LEFT:
        //判断下一个格子是否能够　行
        if(m_nArray > 0 &&
          !pStage->mapData[m_nRow][m_nArray - 1]) {
            b = true;
            break;                                              //撞墙了
        }
        m_ptCenter.x -= m_nSpeed;
        if(m_ptCenter.x < MIN) {
            m_ptCenter.x = MAX;
        }

        break;
    //以下方向的判断原理相同
    case RIGHT:
        if(m_nArray < MAPLENTH - 1 &&
          !pStage->mapData[m_nRow][m_nArray + 1]) {
            b = true;
            break;                                              //撞墙了
        }
        m_ptCenter.x += m_nSpeed;
        if(m_ptCenter.x > MAX) {
            m_ptCenter.x = MIN;
        }
```

```
            break;
        case UP:
            if(m_nRow > 0 &&
              !pStage->mapData[m_nRow - 1][m_nArray]) {
                b = true;
                break;                                    //撞墙了
            }
            m_ptCenter.y -= m_nSpeed;
            if(m_ptCenter.y < MIN) {
                m_ptCenter.y = MAX;
            }
            break;
        case DOWN:
            if(m_nRow < MAPLENTH - 1 &&
              !pStage->mapData[m_nRow + 1][m_nArray]) {
                b = true;
                break;                                    //撞墙了
            }
            m_ptCenter.y += m_nSpeed;
            if(m_ptCenter.y > MAX) {
                m_ptCenter.y = MIN;
            }
            break;
    }
    return b;
}
```

9.8.4　玩家对象的实现

下面是玩家对象的实现代码，这里增加了判断游戏是否胜利的函数 IsWin，判断逻辑是遍历当前地图的数据，查看豆子的数量，如果发现至少还有 1 个豆子，则没有胜利。同时也实现 Draw 方法，该方法负责绘制玩家对象自己，在 GObject.h 文件中继续输入以下代码：

```
//PacMan 成员定义
void PacMan::AchiveCtrl()
{
    GObject::AchiveCtrl();
    if(Achive()) {
        if(m_nRow >= 0 && m_nRow < MAPLENTH &&
          m_nArray >= 0 && m_nArray < MAPLENTH) { //    止数组    界
            if(pStage->peaMapData[m_nRow][m_nArray]) {
                pStage->peaMapData[m_nRow][m_nArray] = false;
            }
        }
    }
}
```

```
void PacMan::action()
{
   Collision();                                    //   行碰撞检测
}
void PacMan::SetTwCommand(TWARDS command)
{
   m_cmd = command;                                //设置移动方向
}

bool PacMan::IsOver()
{
   return m_dir == OVER;                           //判断游戏是否结束
}

bool PacMan::IsWin()
{
   for(int i = 0; i <= MAPLENTH; i++) {
      for(int j = 0; j <= MAPLENTH; j++) {
         if(pStage->peaMapData[i][j] == true) {    //是豆子
            return false;                          //存在任意一个豆子，没取得胜利
         }
      }
   }
   return true;                                    //没有豆子，胜利
}
POINT PacMan::GetPos()
{
   return m_ptCenter;                              //   回对象的中心位置
}

void PacMan::SetOver()
{
   m_dir = OVER;                                   //设置游戏结束
}

void PacMan::Draw(HDC &memDC)
{
   if(m_dir == OVER) {
                                                   //游戏结束，什么也不干
   }
   else if(m_nFrame % 2 == 0) {                    //第 4 帧动画与第 2 帧动画：张嘴形状
      int x1 = 0, x2 = 0, y1 = 0, y2 = 0;
      int offsetX = DISTANCE / 2 + D_OFFSET;       //弧弦交点 X
      int offsetY = DISTANCE / 2 + D_OFFSET;       //弧弦交点 Y
      switch(m_dir) {
         case UP:                                  //向上移动
            x1 = m_ptCenter.x - offsetX;
            x2 = m_ptCenter.x + offsetX;
```

```
        y2 = y1 = m_ptCenter.y - offsetY;
        break;
      case DOWN:                                   //向下移动
        x1 = m_ptCenter.x + offsetX;
        x2 = m_ptCenter.x - offsetX;
        y2 = y1 = m_ptCenter.y + offsetY;
        break;
      case LEFT:                                   //向左移动
        x2 = x1 = m_ptCenter.x - offsetX;
        y1 = m_ptCenter.y + offsetY;
        y2 = m_ptCenter.y - offsetY;
        break;
      case RIGHT:                                  //向右移动
        x2 = x1 = m_ptCenter.x + offsetX;
        y1 = m_ptCenter.y - offsetY;
        y2 = m_ptCenter.y + offsetY;
        break;

    }
        //画出弧型　分
    Arc(memDC, m_ptCenter.x - DISTANCE, m_ptCenter.y - DISTANCE,
      m_ptCenter.x + DISTANCE, m_ptCenter.y + DISTANCE,
      x1, y1,
      x2, y2);
        //画直线　分，最后组合成玩家对象——一个大嘴的形象
    MoveToEx(memDC, x1, y1, NULL);
    LineTo(memDC, m_ptCenter.x, m_ptCenter.y);
    LineTo(memDC, x2, y2);
  }
  else if(m_nFrame % 3 == 0) {                     //第三帧动画：画出整个圆形
    Ellipse(memDC, m_ptCenter.x - DISTANCE, m_ptCenter.y - DISTANCE,
        m_ptCenter.x + DISTANCE, m_ptCenter.y + DISTANCE);
  }
  else {                                           //嘴完全张开的形状
    int x1 = 0, x2 = 0, y1 = 0, y2 = 0;
    switch(m_dir) {
      case UP:                                     //向上移动
        x1 = m_ptCenter.x - DISTANCE;
        x2 = m_ptCenter.x + DISTANCE;
        y2 = y1 = m_ptCenter.y;
        break;
      case DOWN:                                   //向下移动
        x1 = m_ptCenter.x + DISTANCE;
        x2 = m_ptCenter.x - DISTANCE;
        y2 = y1 = m_ptCenter.y;
        break;
      case LEFT:                                   //向左移动
        x2 = x1 = m_ptCenter.x;
        y1 = m_ptCenter.y + DISTANCE;
```

```
            y2 = m_ptCenter.y - DISTANCE;
            break;
        case RIGHT:                                           //向右移动
            x2 = x1 = m_ptCenter.x;
            y1 = m_ptCenter.y - DISTANCE;
            y2 = m_ptCenter.y + DISTANCE;
            break;

        }
            //画出弧形    分
        Arc(memDC, m_ptCenter.x - DISTANCE, m_ptCenter.y - DISTANCE,
            m_ptCenter.x + DISTANCE, m_ptCenter.y + DISTANCE,
            x1, y1,
            x2, y2);
            //画直线    分，最后组合成玩家对象——一个大嘴的形象
        MoveToEx(memDC, x1, y1, NULL);
        LineTo(memDC, m_ptCenter.x, m_ptCenter.y);
        LineTo(memDC, x2, y2);
    }

    m_nFrame++;                                               //绘制下一帧
}
```

9.8.5 完成整个游戏

本节开始使用玩家对象、敌军对象及地图对象来完成整个游戏。

1. 生成游戏窗口

打开 pacman.cpp 文件，找到 wWinMain 函数，将函数内容（大括号之内的内容）删除，输入以下代码：

```
//参数不再使用了
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);

//初始化全局字符串
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_PACMAN, szWindowClass, MAX_LOADSTRING);
//注册窗口类
MyRegisterClass(hInstance);

//执行应用程序初始化
if(!InitInstance(hInstance, nCmdShow)) {
    return FALSE;
}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_PACMAN));
```



2．定义相关变

下面代码定义了当前关卡、3 关地图数组、玩家对象及 4 个敌军对象，并设定当前关卡为第一关。在 pacman.cpp 文件中接着输入：

```
//当前的关卡
int s_n = 0; //[0, 1, 2]
//地图
GMap *MapArray[STAGE_COUNT] = { new Stage_1(), new Stage_2(), new Stage_3() };

//玩家对象
//自己
auto g_me = std::make_shared<PacMan>(P_ROW, P_ARRAY);
//设定 4 个敌人对象
auto e1 = std::make_shared<RedOne>(E_ROW, E_ARRAY);          //红色敌军对象
auto e2 = std::make_shared<RedOne>(E_ROW, E_ARRAY);          //红色敌军对象
auto e3 = std::make_shared<BlueOne>(E_ROW, E_ARRAY);         //蓝色敌军对象
auto e4 = std::make_shared<YellowOne>(E_ROW, E_ARRAY);       //  色敌军对象


//关卡
GObject::pStage = MapArray[s_n];                             //初始化为第一关地图

//设定玩家
Enermy::player = g_me;                                       //用一个指  指向玩家对象

MSG msg;

DWORD dwLastTime = 0;
```

3．游戏循环

在循环中首先判断是否赢得比赛，赢则提示玩家赢得游戏（弹出消息提示框）；在玩家单击“确定”按钮之后，重设玩家和 4 个敌军的状态并进入下一关，如果没有下一关，则跳出循环。接着输入：

```
//主消息循环
//玩家没有被抓，并且关卡小于 3
while(!g_me->IsOver() && s_n < STAGE_COUNT) {
  //判断是否赢得比赛
  if(g_me->IsWin()) {
    s_n++;                                                   //移动到下一关
    //  设自己和敌人位置
    g_me->SetPosition(P_ROW, P_ARRAY);
    e1->SetPosition(E_ROW, E_ARRAY);                         //设置敌军 1 的位置
    e2->SetPosition(E_ROW, E_ARRAY);                         //设置敌军 2 的位置
    e3->SetPosition(E_ROW, E_ARRAY);                         //设置敌军 3 的位置
    e4->SetPosition(E_ROW, E_ARRAY);                         //设置敌军 4 的位置
    //判断是否完成了 3 关，如果完成，   出游戏，否则   入下一关
    if(s_n < 3) {
```

```
            MessageBox(g_hwnd, _T("恭喜　关"), _T("吃豆子提示"), MB_OK);
            GObject::pStage = MapArray[s_n];
            RECT screenRect;
            screenRect.top = 0;
            screenRect.left = 0;
            screenRect.right = WLENTH;
            screenRect.bottom = WHIGHT;

            HDC hdc = GetDC(g_hwnd);                         //获取设备
            std::shared_ptr<HDC__> dc(hdc, [](HDC hdc) {     //智能指　，自动管理资源
                ::ReleaseDC(g_hwnd, hdc);
            });
            ::FillRect(dc.get(), &screenRect, CreateSolidBrush(RGB(255, 255, 255)));
            GObject::pStage->DrawMap(hdc);                   //画地图
            continue;                                        //继续　行循环
        }
        else {
            //　出循环
            break;
        }
    }
```

上述代码判断玩家是否通关，如果通关则进入一下关，但当通过的是第三关时，跳出循环并提示玩家胜利，如果玩家失败，则跳出循环并提示玩家失败。

4．消息处理

输入消息处理部分，此处获取消息的函数为 PeekMessage，该函数不同于 GetMessage 函数，前者无论是否有消息都立即返回，如果有消息，则从队列中移除该消息，而后者则是无消息不返回，一直停在该函数调用处。因此如果使用 GetMessage 函数，无法形成消息循环，导致游戏停止不动，因此这里改用 PeekMessage 函数。

```
//获取消息
if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
    TranslateMessage(&msg);                              //翻译消息
    DispatchMessage(&msg);                               //分发消息
}
```

5．游戏　度调节

若不调节游戏速度，不同计算机的游戏速度会导致游戏运行速度差异较大。为防止这种情况，需要通过时间判断来调节速度。当两次循环的时间没有超过 40 毫秒时，不进行后续的操作，这样就控制了游戏的帧数约为 25 帧/秒。接着输入：

```
//判断时　，否则画得太快
if(GetTickCount() - dwLastTime >= 40) {
    dwLastTime = GetTickCount();                         //记住本次的时
```



```
}
else {
    continue;                                       //时  不到，本次不  行绘画
}
```

6．游戏画　和状态更新

先获得窗口的设备句柄，后面的对象都要画到这个设备上，接着调用地图的方法绘制豆子和地图，再调用对象的成员方法更新状态，绘制玩家对象和敌军对象。最后调用 GetAsyncKeyState 函数，获取按键状态，并设定玩家状态的指令，代码如下：

```
{
    HDC hdc = ::GetDC(g_hwnd);                      //获得设备
    std::shared_ptr<HDC__> dc(hdc, [](auto hdc) {   //不使用时自动  放
        ::ReleaseDC(g_hwnd, hdc);                   //  放设备
    });
    MapArray[s_n]->DrawPeas(hdc);                   //画豆子
    MapArray[s_n]->DrawMap(hdc);                    //画地图

    //画敌人及自动  动
    {
        e1->action();                               //敌军 1 的行为函数
        e1->DrawBlank(hdc);                         //画敌军 1 的空白
        e1->Draw(hdc);                              //画敌军 1 的主体  分

        e2->action();                               //敌军 2 的行为函数
        e2->DrawBlank(hdc);                         //画敌军 2 的空白
        e2->Draw(hdc);                              //画敌军 2 的主体  分

        e3->action();                               //敌军 3 的行为函数
        e3->DrawBlank(hdc);                         //画敌军 3 的空白
        e3->Draw(hdc);                              //画敌军 3 的主体  分

        e4->action();                               //敌军 4 的行为函数
        e4->DrawBlank(hdc);                         //画敌军 4 的空白
        e4->Draw(hdc);                              //画敌军 4 的主体  分
    }

    {
        //画自己
        g_me->DrawBlank(hdc);
        g_me->Draw(hdc);
        //自己向前移动
        g_me->action();

        //获取按  ：控制自己的方向
        if(GetAsyncKeyState(VK_DOWN) & 0x8000) {    //检测到下方向  被按下
            g_me->SetTwCommand(DOWN);               //设置下一步的移动方向为向下
```

```
        }
        if(GetAsyncKeyState(VK_LEFT) & 0x8000) {          //检测到左方向   被按下
            g_me->SetTwCommand(LEFT);                     //设置下一步的移动方向为向左
        }
        if(GetAsyncKeyState(VK_RIGHT) & 0x8000) {         //检测到右方向   被按下
            g_me->SetTwCommand(RIGHT);                    //设置下一步的移动方向为向右
        }
        if(GetAsyncKeyState(VK_UP) & 0x8000) {            //检测到上方向   被按下
            g_me->SetTwCommand(UP);                       //设置下一步的移动方向为向上
        }
      }
    }
}
```

至此，整个消息循环结束，游戏也进入到结束阶段。下文是结束游戏的提醒代码，根据玩家的状态使用消息框 MessageBoxA 进行不同的提示。

```
//如果游戏结束
if(g_me->IsOver()) {
    MessageBoxA(NULL, "出师未捷", "吃豆子提示", MB_OK);
}
//否则，提示赢得游戏
else {
    MessageBoxA(NULL, "恭喜您赢得了胜利\r\n 确定后游戏   出", "吃豆子提示", MB_OK);
}

return (int) msg.wParam;
```

运行程序，效果如 9.27 所示。

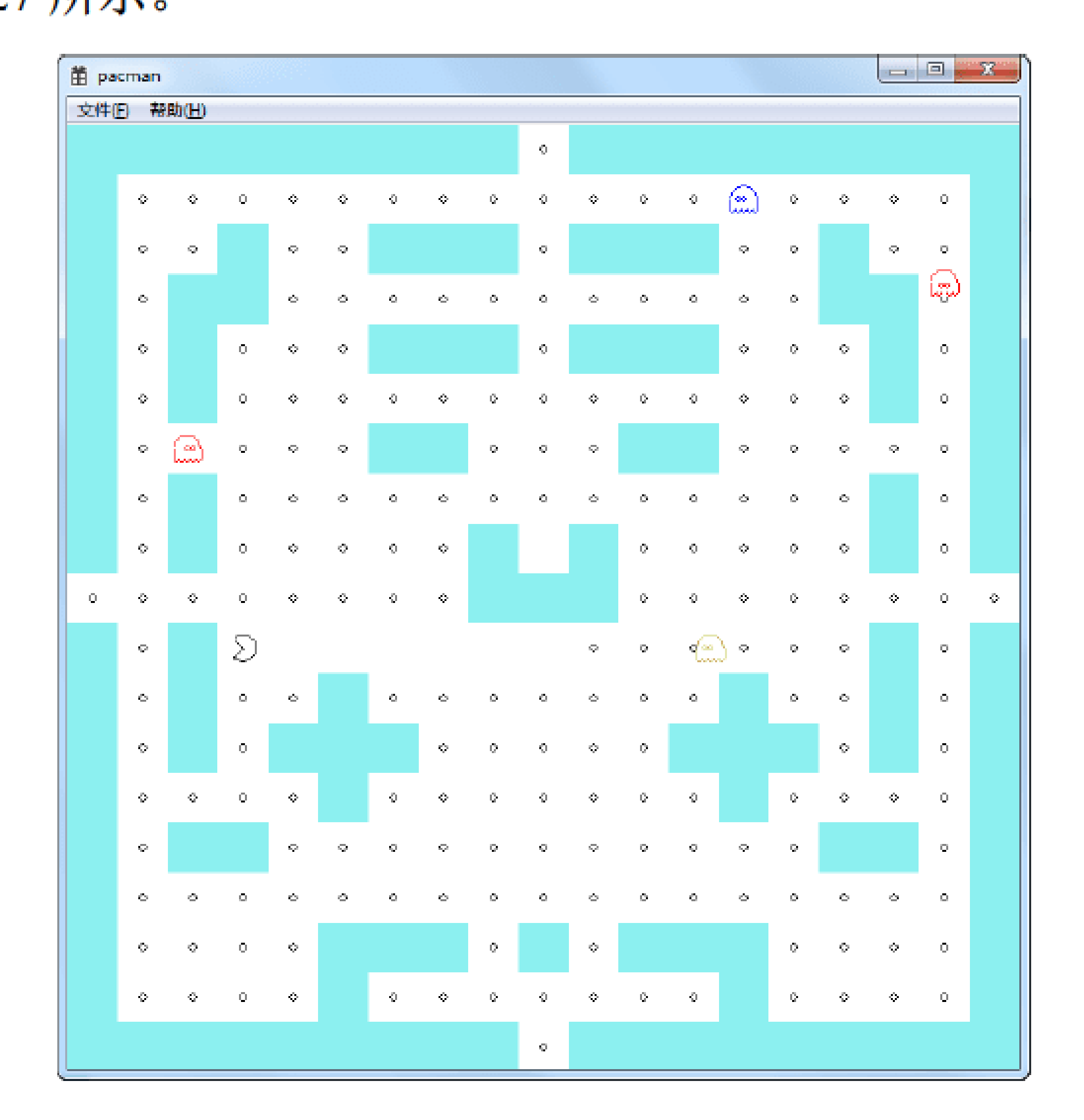
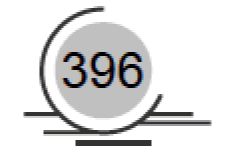


图 9.27　游戏运行效果





9.9　项目文件清单

吃豆子游戏的文件清单如表 9.1 所示。

表 9.1　吃豆子游戏文件清单

文　件　名	文 件 类 型	说　　明
GMap.h	头文件	地图类的声明文件
GMap.cpp	源文件	地图类的实现文件
GObject.h	头文件	物体类的声明文件
GObject.cpp	源文件	物体类的实现文件
pacman.cpp	源文件	创建主窗口，实现游戏运行的客户端

9.10　本 章 总 结

本章设计了一个吃豆子游戏，目的是让读者理解类的多态与继承的使用方法。Windows API 内容很多，完全掌握需要大量时间和实践，不作为本章的重点。地图类与物体类使用了 C++类继承的特性，子类的共性应该放入父类中，性质相似而实现不同的函数则应该由虚函数定义。

第10章

快乐五子棋

（Visual Studio 2017+Socket 套接字实现）

五子棋是起源于中国古代的传统黑白棋种之一。五子棋不仅能增强思维能力，提高智力，而且富含哲理，有助于修身养性，既有现代休闲游戏的明显特征“短、平、快”，又有古典哲学的高深学问“阴、阳、易、理”；既具有简单易学的特性，为人们所喜爱，又有深奥的技巧和高水平的国际性比赛。五子棋文化源远流长，具有东方的神秘和西方的直观；既有“场”的概念，亦有“点”的连接，起源于中国古代，发展于日本，风靡于欧洲，可以说五子棋是中西方文化的交流点，是古今哲学的结晶。在本章中，笔者将设计一个网络五子棋游戏。

通过学习本章，读者可以学到：

- 使用 TCP 协议进行网络通信
- 定义网络通信协议
- 绘制可以调整大小的五子棋棋盘
- 在棋盘上绘制棋子，在窗口更新时能够保留绘制的棋子
- 五子棋赢棋判断
- 游戏回放
- 网络连接状态检测
- 向对话框中嵌入子对话框，并实现动态调整子对话框的大小


配置说明

10.1　开 发 背 景



相信很多人都会玩五子棋游戏，当一方完成 5 个棋子连续时，无论是水平方向、垂直方向，还是斜对角线方向，都表示获胜了。对于初学网络开发的人员来说，设计一个网络五子棋游戏再合适不过了。从规模上看，网络五子棋只需要包含客户端和服务器端两个窗口，规模比较小。从功能上看，网络五子棋涉及两台主机间的通信，需要相互传递棋子信息、控制指令和文本信息，这需要定义一个应用协议来解释数据报，涉及网络开发的许多知识。鉴于此，在本章中笔者设计了一个网络五子棋模块。

10.2　需 求 分 析

通过调查，要求系统具有以下功能。

☑ 为了体现良好的娱乐性，因此要求系统具有良好的人机交互界面。
☑ 完全人性化设计，无须专业人士指导即可操作本系统。
☑ 自动完成胜负判断，避免人为错误。
☑ 实现游戏悔棋。
☑ 实现游戏回放。
☑ 实现游戏双方的网络通话。

10.3　系 统 设 计

10.3.1　系统功能结构

快乐五子棋游戏包括客户端和服务器端两个应用程序，系统功能结构图如图 10.1 所示。

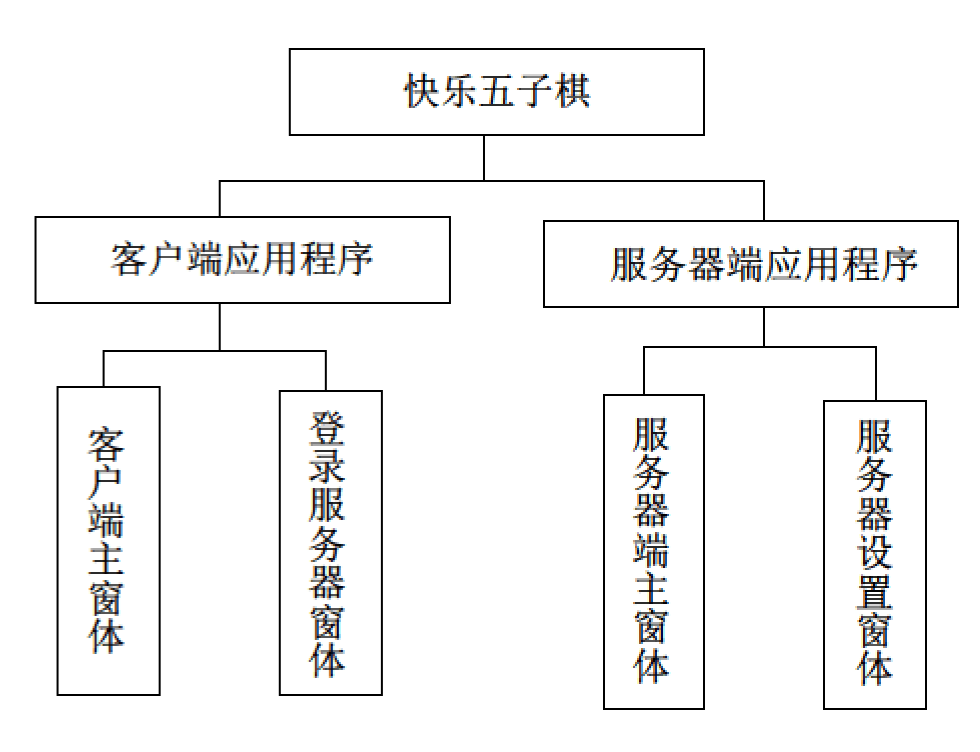


图 10.1　快乐五子棋结构图

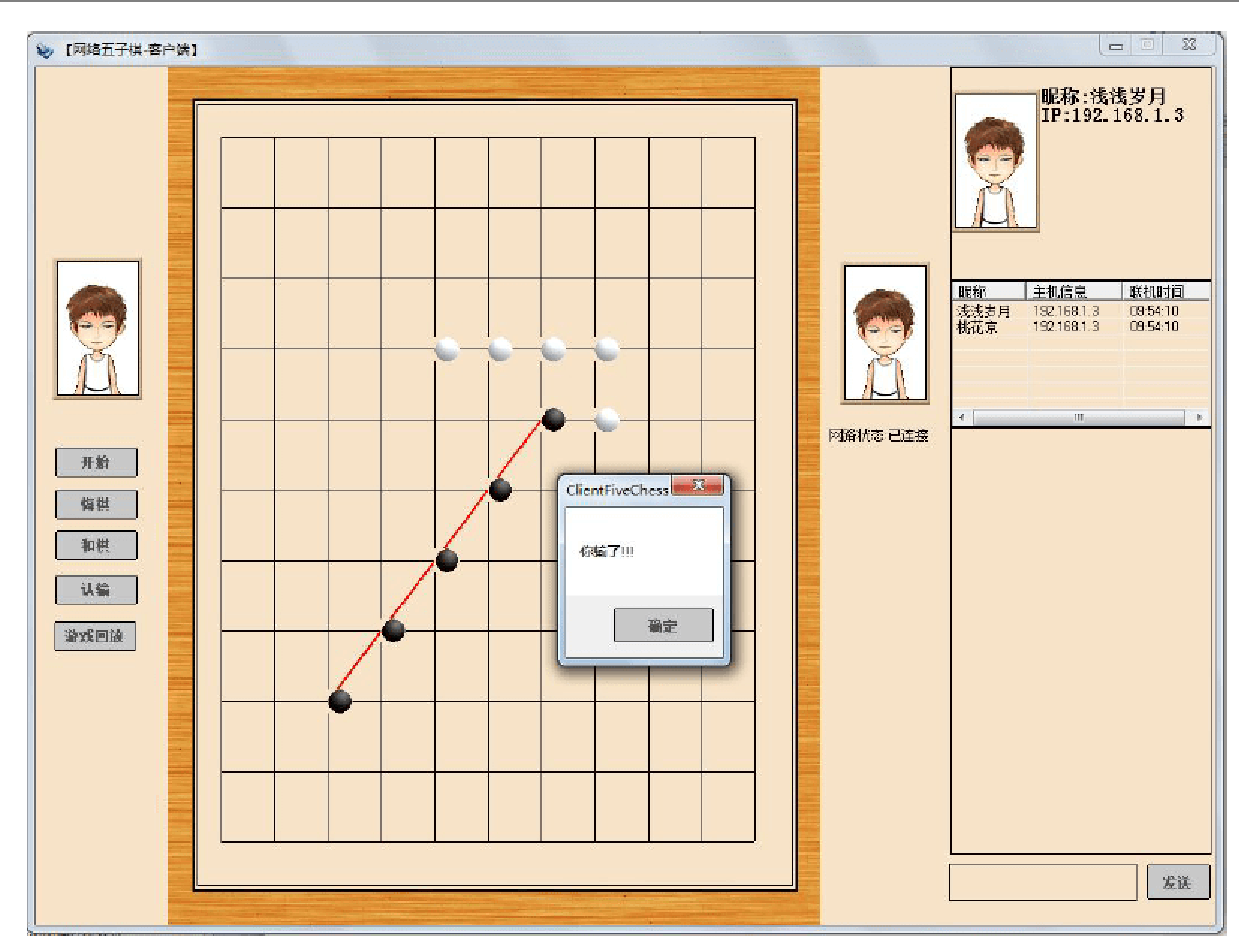
10.3.2　系统预览

系统预览分为客户端预览和服务器端预览两部分，客户端主要由一个主窗体和一个登录服务器窗体构成；服务器端由一个主窗体和一个服务器设置窗体构成。客户端主窗体效果如图 10.2 所示。

登录服务器窗体效果如图 10.3 所示。

服务器端主窗体效果如图 10.4 所示。

服务器设置窗体如图 10.5 所示。



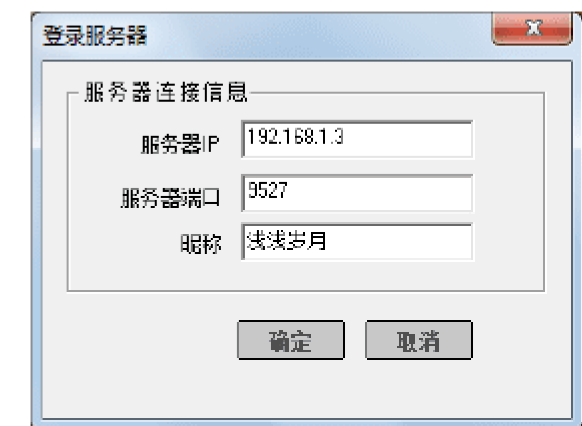
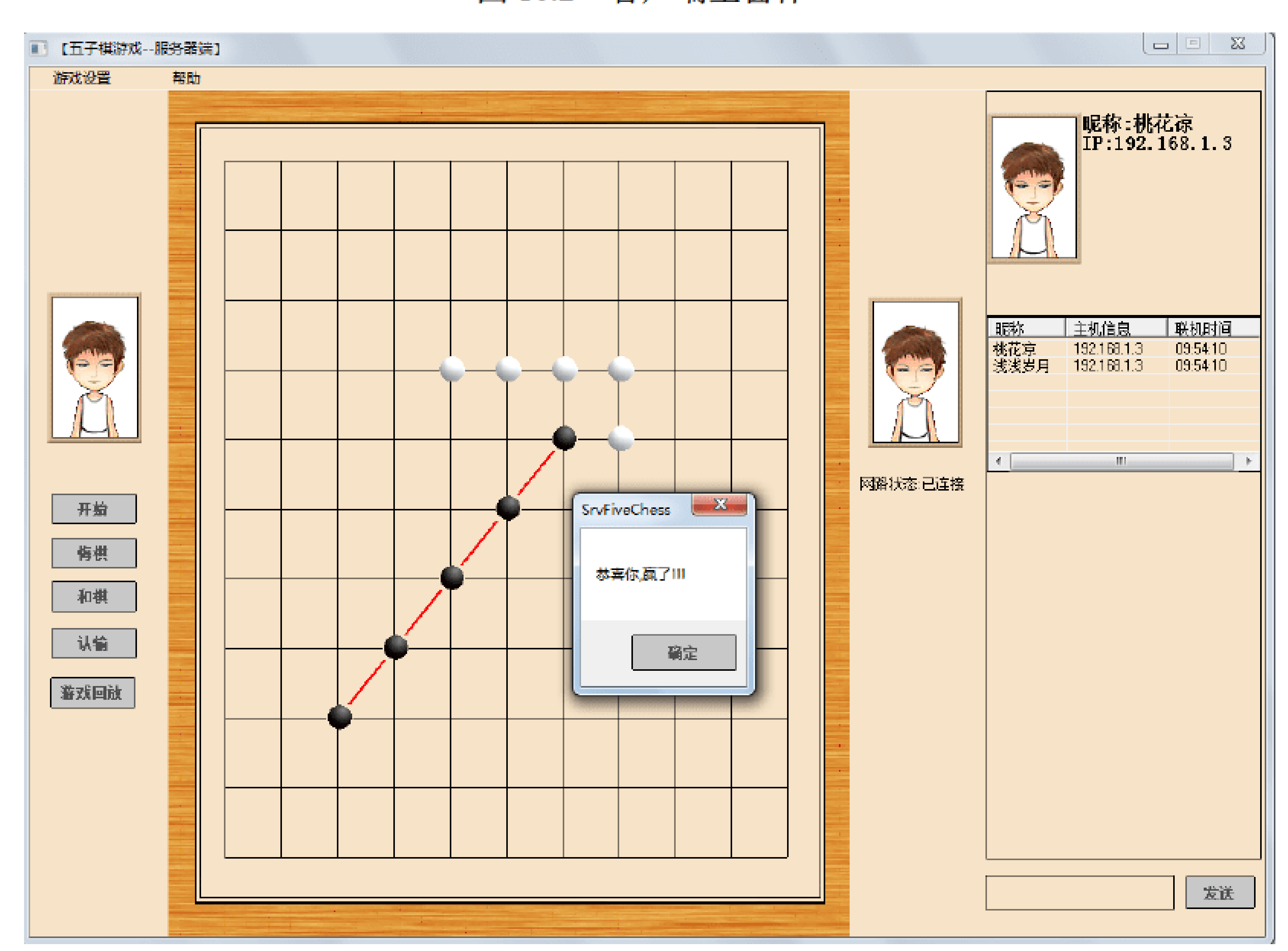


图 10.2　客户端主窗体　　　　图 10.3　登录服务器窗体



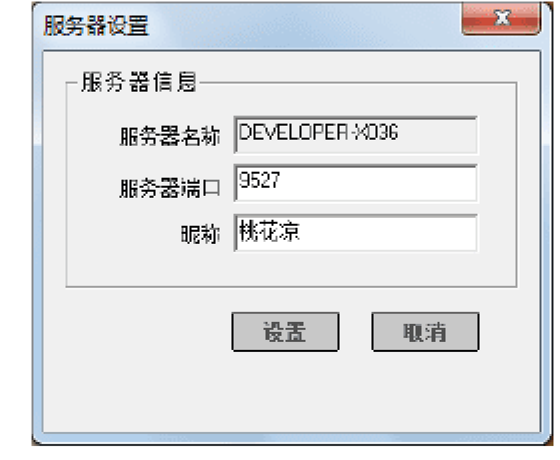


图 10.4　服务器端主窗体　　　　图 10.5　服务器设置窗体

10.3.3　业务流程图

快乐五子棋业务流程图如图 10.6 所示。

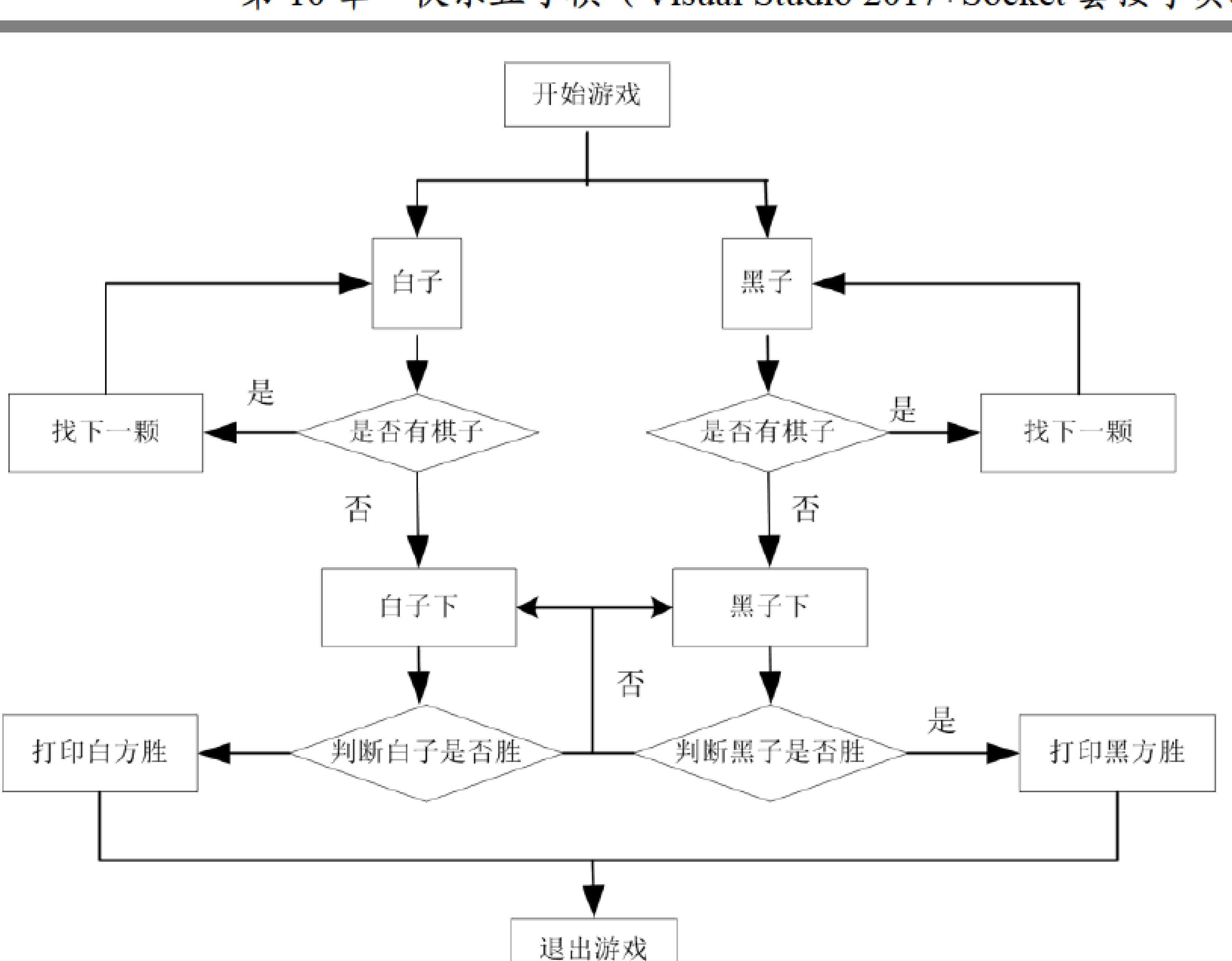


图 10.6　快乐五子棋业务流程图

10.3.4　程序运行环境

本系统对其运行环境有一定的要求，具体如下。

☑　系统开发平台：Visual Studio 2017。
☑　系统开发语言：C++。
☑　数据库管理系统软件：Microsoft Access 2000。
☑　运行平台：Windows XP（SP2）/Windows 2000（SP4）/Windows Server 2003（SP1）/ Windows 7。
☑　分辨率：最佳效果为 1024×1280 像素。

10.4　关键技术分析与实现



10.4.1　使用 TCP 进行网络通信

TCP（Translate Control Protocol，传输控制协议）提供了一个完全可靠的、面向连接的、全双工的字节流传输服务。在设计网络五子棋模块时，考虑到网络传输的数据量不是很大，要求数据准确地传递给对方，笔者决定使用 TCP 进行网络通信。

采用 TCP 进行网络通信的编程模式为：首先创建一个 TCP 套接字，然后将套接字绑定到本机的 IP 和端口号上，之后将套接字置于监听模式，当有客户端的套接字连接时，接收客户端的连接请求，这样双方就可以进行通信了。在 Visual Studio 2017 中，可以采用两种方式来进行套接字编程，一种方式

是使用套接字的 API 函数，另一种方式是使用 MFC 提供的套接字类 CAsyncSocket 和 CSocket。在本模块中采用第二种方式——使用 CSocket 类进行网络通信。下面介绍使用 CSocket 类进行网络编程的基本步骤。

（1）从 CSocket 类派生一个子类，如 CSrvSock。

（2）改写 CSocket 类的 OnAccept 方法，当有客户端连接时，调用自定义的方法来接收连接。

```
void CSrvSock::OnAccept(int nErrorCode)
{
	m_pDlg->AcceptConnection();                    //在主对话框中自定义的方法，用于接收客户端连接
	CSocket::OnAccept(nErrorCode);
}
//自定义的 AcceptConnection 方法，用于接收客户端的连接
void CChessBorad::AcceptConnection()
{
	m_ClientSock.Close();                          //关闭套接字
	m_SrvSock.Accept(m_ClientSock);                //接收连接
}
```

（3）从 CSocket 类再派生一个子类，如 CClientSock。

（4）改写 CSocket 类的 OnReceive 方法，当客户端发送数据时，将调用自定义的方法接收数据。

```
void CClientSock::OnReceive(int nErrorCode)
{
	if (m_pDlg != NULL)
		m_pDlg->ReceiveData();                     //调用主对话框自定义方法，接收数据
	CSocket::OnReceive(nErrorCode);
}
```

自定义的 ReceiveData 方法，当服务器端检测到客户端发送的数据时接收数据。

```
void CChessBorad::ReceiveData()
{
	BYTE* pBuffer = new BYTE[sizeof(TCP_PACKAGE)];     //定义一个缓冲区
	//接收客户端发来的数据
	int factlen = m_ClientSock.Receive(pBuffer,sizeof(TCP_PACKAGE));
	delete []pBuffer;                              //释放缓冲区
}
```

（5）在 StdAfx.h 头文件中引用 afxsock.h 头文件，目的是使用 CSocket 类。

```
#include <afxsock.h>
```

（6）在应用程序初始化时调用 AfxSocketInit 方法初始化套接字函数库。

```
WSADATA wsa;
AfxSocketInit(&wsa);                               //初始化套接字
```

至此就完成了对套接字类 CSocket 的封装。下面通过代码来说明套接字类的通信过程。



（1）创建并绑定套接字地址和端口。

```
m_SrvSock.Create(port,SOCK_STREAM,SrvDlg.m_HostIP);          //创建套接字
```

Create 方法的第二个参数为 SOCK_STREAM，表示创建 TCP 套接字。

（2）将套接字置于监听模式。

```
m_SrvSock.Listen();                                          //监听套接字
```

（3）在客户端创建并绑定套接字地址和端口。

```
m_ClientSock.Create();                                       //创建客户端套接字
```

（4）客户端套接字开始连接服务器套接字。

```
m_ClientSock.Connect(srvDlg.m_IP,srvDlg.m_Port);             //连接服务器
```

此时服务器端的套接字将调用 OnAccept 方法（CSrvSock 类），执行自定义的 AcceptConnection 方法接收客户端的连接，这样，客户端就可以和服务器端进行通信了。例如，向服务器发送一行文本数据，代码如下：

```
m_ClientSock.Send("明日科技",8);                             //向服务器发送数据
```

10.4.2　定义网络通信协议

在设计网络应用程序时，通常需要定义一个应用协议，使通信双方按照此协议来解释接收的数据。以网络五子棋模块为例，网络通信的数据主要包括文本数据、开始游戏命令、网络测试命令、五子棋坐标、悔棋请求命令、接受悔棋请求命令和拒绝悔棋请求命令等。这些类型的数据需要在接收端按照预定的协议解析出来，然后执行相应的动作。下面给出网络五子棋模块定义的通信协议。

```
/*********************枚举常量说明**************************
CT_BEGINGAME                    开始游戏
CT_NETTEST                      网络测试
CT_POINT                        棋子坐标
CT_TEXT                         文本信息
CT_WINPOINT                     赢棋时的起点和终点棋子
CT_BACKREQUEST                  悔棋请求
CT_BACKACCEPTANCE               同意悔棋
CT_BACKDENY                     拒绝悔棋
CT_DRAWCHESSREQUEST             和棋请求
CT_DRAWCHESSACCEPTANCE          同意和棋
CT_DRAWCHESSDENY                拒绝和棋
CT_GIVEUP                       认输
************************************************************************/
enum CMDTYPE  { CT_BEGINGAME,CT_NETTEST,CT_POINT,CT_TEXT,CT_WINPOINT,
                CT_BACKREQUEST,CT_BACKACCEPTANCE,CT_BACKDENY,
                CT_DRAWCHESSREQUEST,CT_DRAWCHESSACCEPTANCE,
                CT_DRAWCHESSDENY,CT_GIVEUP
```

```
        };
//定义数据包结构
struct TCP_PACKAGE
{
    CMDTYPE cmdType;                                    //命令类型
    CPoint chessPT;                                     //五子棋坐标（行和列坐标）
    CPoint winPT[2];                                    //赢棋时的路径（起点和终点）
    char chText[512];                                   //文本数据
};
```

在定义了通信协议后，通信双方在发送数据时，需要按照数据的类型填充数据包。例如，要发送开始游戏的请求，需要按照如下的格式填充数据包：

```
//发送游戏开始的信息
TCP_PACKAGE tcpPackage;                                 //定义数据包格式
memset(&tcpPackage,0,sizeof(TCP_PACKAGE));              //初始化数据包
tcpPackage.cmdType = CT_BEGINGAME;                      //设置命令类型
strncpy(tcpPackage.chText,m_csNickName,512);            //设置昵称
m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));     //发送数据包
```

这样，当对方接收到数据包时，会根据数据包的类型来执行相应的动作。

```
BYTE* pBuffer = new BYTE[sizeof(TCP_PACKAGE)];          //定义缓冲区
//从套接字中读取数据
int factlen = m_ClientSock.Receive(pBuffer,sizeof(TCP_PACKAGE));
if (factlen == sizeof(TCP_PACKAGE))                     //判断读取数据的大小
{
    TCP_PACKAGE tcpPackage;                             //定义一个数据包
//复制缓冲区数据到数据包中
❶       memcpy(&tcpPackage,pBuffer,sizeof(TCP_PACKAGE));
    if (tcpPackage.cmdType == CT_BEGINGAME)             //开始游戏
    {
            //进行游戏开始的操作
    }
}
```

代码贴士

❶ memcpy：用于复制缓冲区中的内容。将代码 pBuffer 中的内容复制到 tcpPackage 中，复制内容大小由 sizeof(TCP_PACKAGE)来确定。

10.4.3 实现动态调整棋盘大小

在设计网络五子棋时，为了突出游戏的特点，允许用户在游戏进行的过程中调整窗口的大小，效果如图 10.7 和图 10.8 所示。



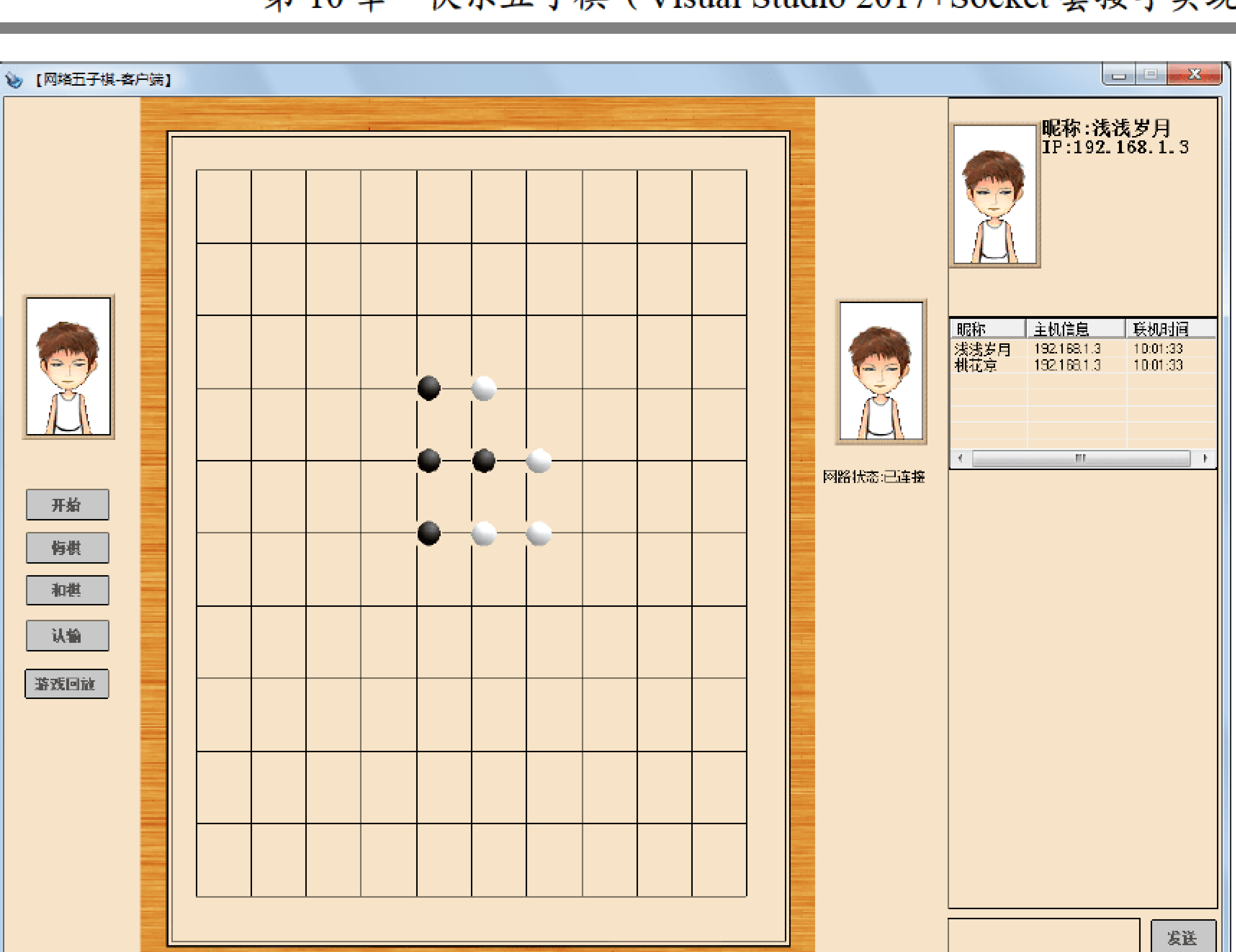


图 10.7　客户端窗口 1

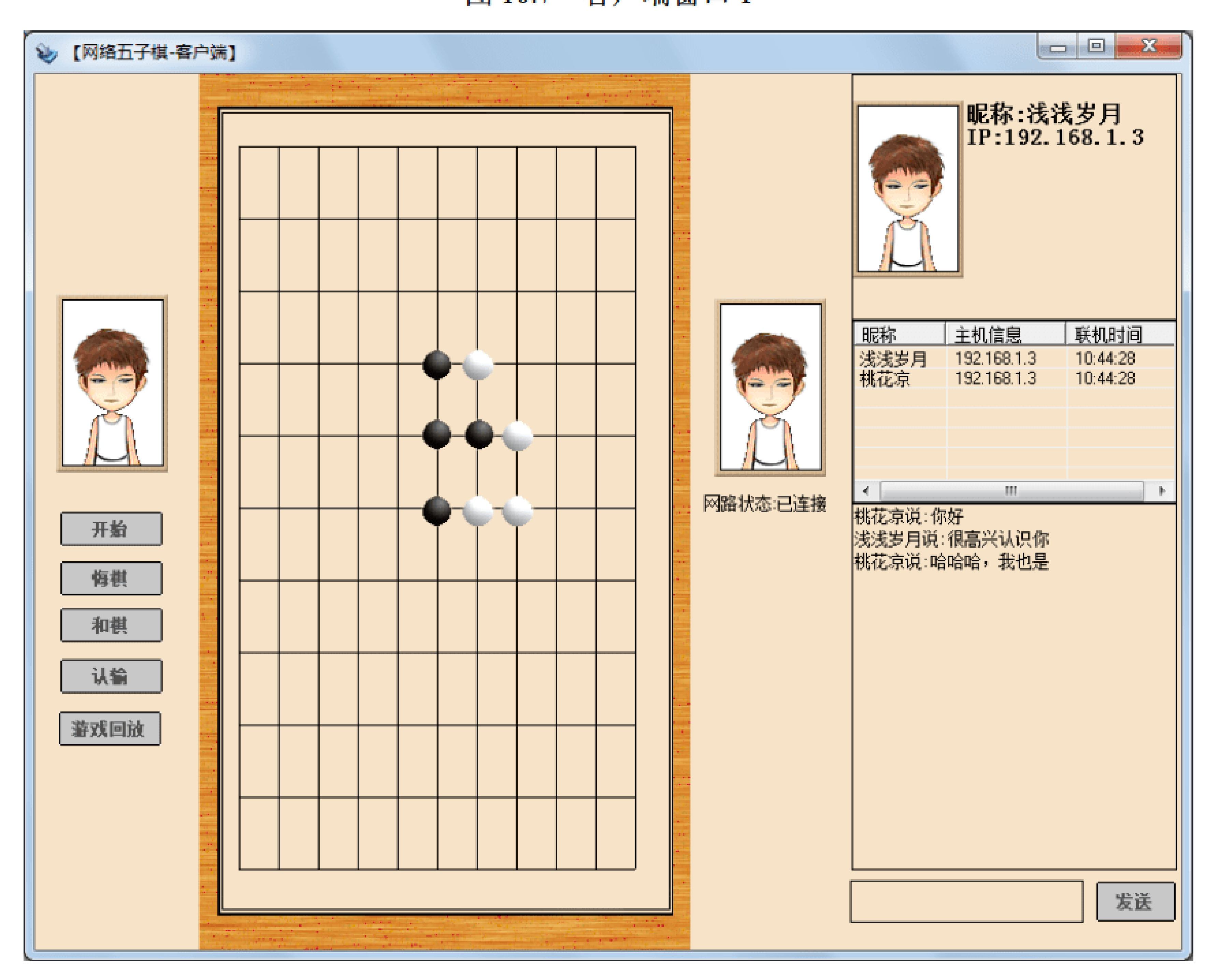
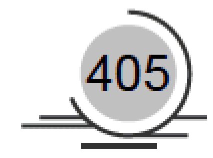


图 10.8　客户端窗口 2



实现该功能的难点在于窗口大小被调整后，需要调整棋盘的大小、棋盘表格的大小和棋盘中当前棋子的位置。此处采用的方式是记录水平方向和垂直方向的缩放比例，当首次显示窗口时，认为水平方向和垂直方向的缩放比例为 1，并记录棋盘的宽度和高度，作为棋盘的原始宽度和高度。当调整窗口时，设置棋盘新的宽度和高度，并将其与原始棋盘的宽度和高度进行除法运算，记录水平方向和垂直方向的缩放比例。在绘制棋盘表格和棋子位置时都依据缩放比例进行绘制。

以绘制棋盘的表格为例，在窗口初始化时需要确定表格相对棋盘的坐标以及表格中每个单元格的宽度和高度。以本模块为例，首次绘制表格时，起点坐标分别为 50 和 50，单元格的宽度和高度均为 50。

```
m_nOrginX = m_nOrginY = 50;                      //表格起点坐标
m_nCellHeight = m_nCellWidth = 50;               //单元格高度和宽度
```

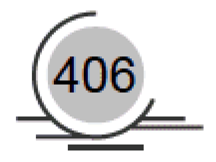
绘制表格时，系统会根据当前水平方向和垂直方向的缩放比例计算此刻表格的起点坐标、单元格的高度和宽度，这样就可以正确地绘制表格了。

```
void CChessBorad::DrawChessboard()
{
    CDC* pDC = GetDC();                                 //获取窗口设备上下文
    CPen pen(PS_SOLID,1,RGB(0,0,0));                    //定义黑色的画笔
    pDC->SelectObject(&pen);                            //选中画笔
    int nOriginX = m_nOrginX*m_fRateX;                  //计算表格的起点坐标
    int nOriginY = m_nOrginY*m_fRateY;
    int nCellWidth = m_nCellWidth*m_fRateX;             //计算单元格的宽度和高度
    int nCellHeight = m_nCellHeight*m_fRateY;
    for (int i = 0; i<m_nRowCount+1; i++)               //绘制棋盘中的列
    {
        pDC->MoveTo(nOriginX+nCellWidth*(i),nOriginY);
        pDC->LineTo(nOriginX+nCellWidth*(i),nOriginY+m_nRowCount*nCellHeight);
    }
    for (int j = 0; j<m_nColCount+1; j++)               //绘制棋盘中的行
    {
        pDC->MoveTo(nOriginX ,nOriginY+(j)*nCellHeight);
        pDC->LineTo(nOriginX +m_nColCount*nCellWidth,nOriginY+(j)*nCellHeight);
    }
}
```

10.4.4 在棋盘中绘制棋子

在设计网络五子棋时，需要在棋盘中绘制棋子，并保证在窗口更新时，棋子仍然在棋盘上。此处采用的方式是定义一个二维数组，数组的大小与棋盘中表格的行和列有关，用以描述棋盘中可以放置的所有棋子。每一个棋子关联一个数据结构 NODE，定义如下：

```
//定义节点颜色
/*************************************************************************
ncWHITE:      表示白色棋子
ncBLACK:      表示黑色棋子
```



```
ncUNKOWN:          表示棋子颜色未知，当没有在棋盘中放置棋子时，棋子为 ncUNKOWN
**********************************************************************/
typedef enum NODECOLOR{ ncWHITE,ncBLACK,ncUNKOWN};
//定义节点类
class NODE
{
public:
    NODECOLOR m_Color;                                    //棋子颜色
    CPoint m_Point;                                       //棋子的物理坐标点
    int m_nX;                                             //棋子的逻辑横坐标
    int m_nY;                                             //棋子的逻辑纵坐标
public:
    NODE* m_pRecents[8];                                  //临近棋子
    BOOL m_IsUsed;                                        //棋子是否被用
    NODE()
    {
        m_Color = ncUNKOWN;
        m_IsUsed = FALSE;
    }
    ~NODE()
    {
    }
};
```

当用户在棋盘中放置一个棋子时，会根据鼠标的坐标点从一个二维数组中获取对应的一个棋子。如果该棋子没有被使用，则设置棋子的颜色，并将棋子标记为已用。这样，在窗口更新时，从二维数组中遍历棋子，如果棋子已用，则根据棋子的坐标和颜色绘制棋子。

```
for (int m=0; m< m_nRowCount+1; m++)                      //遍历行
{
    for (int n=0; n<m_nColCount+1; n++)                   //遍历列
    {
        if (m_NodeList[m][n].m_Color == ncWHITE)          //如果为白色棋子
        {
            memDC.SelectObject(&BmpWhite);                //选中白色棋子位图
            pDC->StretchBlt(m_NodeList[m][n].m_Point.x-nPosX,
                m_NodeList[m][n].m_Point.y-nPosY,nBmpWidth,nBmpHeight,
                &memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY); //绘制白色棋子
        }
        else if (m_NodeList[m][n].m_Color == ncBLACK)     //如果为黑色棋子
        {
            memDC.SelectObject(&BmpBlack);                //选中黑色棋子位图
            pDC->StretchBlt(m_NodeList[m][n].m_Point.x-nPosX,
                m_NodeList[m][n].m_Point.y-nPosY,nBmpWidth,nBmpHeight,
                &memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY); //绘制黑色棋子
        }
    }
}
```

这里还涉及一个问题：如何根据鼠标的坐标点从二维数组中获取对应的棋子？此处采用的方式是在窗口初始化时为每一个棋子设置一个坐标点。采用这种方式是因为在绘制表格时知道表格的起点坐标、单元格的宽度和高度，自然可以知道每一个表格交叉点的坐标，也就是棋子的坐标。

```
for (int i=0; i<m_nRowCount+1; i++)                             //遍历行
{
    for (int j=0; j<m_nColCount+1; j++)                         //遍历列
    {
        //设置节点的坐标
        m_NodeList[i][j].m_Point= CPoint(nOriginX+nCellWidth*j,nOriginY+nCellHeight*i);
    }
}
```

当窗口大小改变时，还将根据缩放比例重新设置每一个棋子的坐标。

```
void CChessBorad::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
    //当窗口大小改变时确定图像的缩放比例
    CRect cltRC;
    GetClientRect(cltRC);                                        //获取窗口客户区域
    m_fRateX = cltRC.Width() / (double)m_nBmpWidth;              //计算新的缩放比例
    m_fRateY = cltRC.Height() / (double)m_nBmpHeight;
    int nOriginX = m_nOrginX*m_fRateX;                           //计算表格新的起点坐标
    int nOriginY = m_nOrginY*m_fRateY;
    int nCellWidth = m_nCellWidth*m_fRateX;                      //计算表格单元格新的宽度和高度
    int nCellHeight = m_nCellHeight*m_fRateY;
    for (int i=0; i<m_nRowCount+1; i++)                          //重新设置棋子的坐标
    {
        for (int j=0; j<m_nColCount+1; j++)
        {
            m_NodeList[i][j].m_Point= CPoint(nOriginX+nCellWidth*j,nOriginY+nCellHeight*i);
        }
    }
}
```

知道了每个棋子的坐标，根据鼠标的坐标点就可以获取对应的棋子坐标。但是在判断坐标时，还需要设置一个近似的区域。例如，棋子的坐标为（100,80），而鼠标的坐标为（98,87），如果进行精确比较，则在鼠标点处获取不到棋子，玩家也不可能准确地单击到棋子坐标。为此需要进行一个近似比较。这里采用的方式是以棋子坐标为中心点，设置一个区域，只要鼠标点位于该区域中，则返回该棋子坐标。

```
//根据坐标点获取棋子
NODE* CChessBorad::GetLikeNode(CPoint pt)
{
    CPoint tmp;
    for (int i = 0 ;i<m_nRowCount+1;i++)                         //遍历行
        for (int j = 0; j<m_nColCount+1;j++)                     //遍历列
```

```
        {
            tmp = m_NodeList[i][j].m_Point;                     //获取棋子坐标
            int nSizeX = 10 * m_fRateX;
            int nSizeY = 10 * m_fRateY;
            //定义一个临近棋子的区域
            CRect rect(tmp.x-nSizeX,tmp.y-nSizeY,tmp.x+nSizeX,tmp.y+nSizeY);
            if (rect.PtInRect(pt))                              //判断鼠标指针是否在临近区域
                return &m_NodeList[i][j];                       //返回棋子坐标
        }
    return NULL;
}
```

10.4.5　五子棋赢棋判断

在设计五子棋模块时，需要提供一个算法判断用户或者对方是否赢棋。根据五子棋规则，只要在水平方向、垂直方向或两个对角线方向的任意一个方向存在 5 个连续颜色的棋子即表示获胜。为了能够进行赢棋判断，在设计棋子结构时，定义一个 m_pRecents[8]成员，该成员表示当前节点周围的临近节点，如图 10.9 所示。

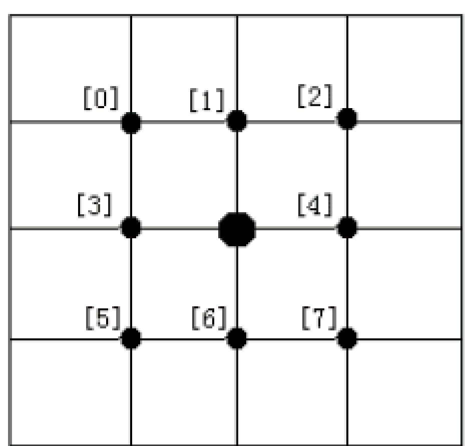


图 10.9　棋子的 8 个临近节点示意图

一些位于表格边缘的棋子是没有 8 个临近节点的，则该棋子的某些临近节点为空。下面的代码用于为某一个棋子设置临近节点。

```
void CChessBorad::SetRecentNode(NODE *pNode)
{
    int nCurX = pNode->m_nX;                                    //获取当前节点的行索引
    int nCurY = pNode->m_nY;                                    //获取当前节点的列索引
    if (nCurX > 0 && nCurY >0)                                  //左上方临近节点
        pNode->m_pRecents[0] = &m_NodeList[nCurX-1][nCurY-1];
    else
        pNode->m_pRecents[0] = NULL;
    if (nCurY > 0)                                              //上方临近节点
        pNode->m_pRecents[1] = &m_NodeList[nCurX][nCurY-1];
    else
        pNode->m_pRecents[1] = NULL;
    if (nCurX < m_nColCount-1 && nCurY > 0)                     //右上方临近节点
        pNode->m_pRecents[2] = &m_NodeList[nCurX+1][nCurY-1];
    else
```

```
        pNode->m_pRecents[2] = NULL;
    if (nCurX >0)                                          //左方临近节点
        pNode->m_pRecents[3] = &m_NodeList[nCurX-1][nCurY];
    else
        pNode->m_pRecents[3] = NULL;
    if (nCurX < m_nColCount-1)                             //右方临近节点
        pNode->m_pRecents[4] = &m_NodeList[nCurX+1][nCurY];
    else
        pNode->m_pRecents[4] = NULL;
    if (nCurX >0 && nCurY < m_nRowCount-1)                 //左下方临近节点
        pNode->m_pRecents[5] = &m_NodeList[nCurX-1][nCurY+1];
    else
        pNode->m_pRecents[5] = NULL;
    if (nCurY < m_nRowCount-1)                             //下方临近节点
        pNode->m_pRecents[6] = &m_NodeList[nCurX][nCurY+1];
    else
        pNode->m_pRecents[6] = NULL;
    if (nCurX < m_nColCount-1 && nCurY < m_nRowCount-1)    //右下方临近节点
        pNode->m_pRecents[7] = &m_NodeList[nCurX+1][nCurY+1];
    else
        pNode->m_pRecents[7] = NULL;
}
```

如果为每个棋子都设置了临近节点，那么通过一个棋子就可以遍历整个棋盘中的所有节点，也就可以判断五子棋的输赢了。在判断五子棋输赢时需要从水平方向、垂直方向和对角线方向分别进行判断。以从垂直方向判断为例，以当前棋子为中心，向上方查找同一个颜色的棋子，有则累加计数，然后从当前节点的下方开始查找节点，如果有同一个颜色的棋子，则继续累加计数，当计数达到 5 时，则表示赢棋。赢棋判断的代码如下：

```
NODE* CChessBorad::IsWin(NODE *pCurrent)
{
    if (pCurrent->m_Color != ncBLACK)
        return NULL;
    //按 4 个方向判断
    int num = 0;                                           //定义计数
    m_Startpt.x = pCurrent->m_nX;
    m_Startpt.y = pCurrent->m_nY;
    m_Endpt.x = pCurrent->m_nX;
    m_Endpt.y = pCurrent->m_nY;
    //按垂直方向判断，在当前节点分别按上、下两个方向遍历
    NODE* tmp = pCurrent->m_pRecents[1];                   //获得当前节点的上方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color) //遍历上方节点
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                          //累加连续棋子数量
        if (num >= 4)                                      //是否有 5 个连续棋子
        {
```

```
            return tmp;                                 //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[1];
    }
    tmp = pCurrent->m_pRecents[6];                      //获得当前节点的下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)  //遍历下方节点
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;
        if (num >= 4)
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[6];
    }
    //按水平方向判断，在当前节点处分别向左、右两个方向遍历
    num = 0;
    tmp = pCurrent->m_pRecents[3];                      //遍历左节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                       //累加连续棋子数量
        if (num >= 4)                                   //是否有 5 个连续棋子
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[3];
    }
    tmp = pCurrent->m_pRecents[4];                      //遍历右节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;
        if (num >= 4)
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[4];
    }
    num = 0;
    //按 135° 斜角遍历
    tmp = pCurrent->m_pRecents[0];
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)  //遍历斜上方节点
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
```

```
        num += 1;                                          //累加连续棋子数量
        if (num >= 4)                                      //是否有 5 个连续棋子
        {
            return tmp;                                    //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[0];
    }
    tmp = pCurrent->m_pRecents[7];                         //遍历斜下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;                                          //累加连续棋子数量
        if (num >= 4)                                      //是否有 5 个连续棋子
        {
            return tmp;                                    //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[7];
    }
    //按 45° 斜角遍历
    num = 0;
    tmp = pCurrent->m_pRecents[2];                         //遍历斜上方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                          //累加连续棋子数量
        if (num >= 4)                                      //是否有 5 个连续棋子
        {
            return tmp;                                    //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[2];
    }
    tmp = pCurrent->m_pRecents[5];                         //遍历斜下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;                                          //累加连续棋子数量
        if (num >= 4)                                      //是否有 5 个连续棋子
        {
            return tmp;                                    //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[5];
    }
    return NULL;
}
```

10.4.6　设计游戏悔棋功能

为了增加网络五子棋的灵活性，在本模块中添加了悔棋功能。当用户想要悔棋时，需要向对方发送悔棋请求，如果对方同意悔棋，则双方都进行悔棋操作；如果对方不同意悔棋，则向发送请求的一方发出拒绝悔棋消息。

为了实现悔棋功能，在客户端和服务器端都定义了两个成员变量，分别记录当前用户最近放置棋子的逻辑坐标和对方最近放置棋子的逻辑坐标。实现悔棋的效果处理非常简单，只需要将最近放置的棋子的颜色设置为 ncUNKOWN，然后重绘窗口即可。这里有一个问题需要注意，在进行悔棋时，如果轮到本地用户下棋时进行悔棋操作，需要撤销两个棋子，第一个棋子是之前对方放置的棋子，第二个棋子是之前本地用户放置的棋子。如果轮到对方下棋时进行悔棋，则只需要撤销一个棋子，即之前本地用户放置的棋子。下面给出实现悔棋功能的主要代码。

（1）发送悔棋请求，代码如下：

```
void CLeftPanel::OnBtBack()
{
    CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
    if (pDlg->m_ChessBoard.m_State==esBEGIN)                //判断游戏是否正在进行
    {
        //发出悔棋请求
        TCP_PACKAGE tcpPackage;                             //定义数据包
        tcpPackage.cmdType = CT_BACKREQUEST;                //设置数据包命令类型
        //用户已经下棋
        if (pDlg->m_ChessBoard.m_LocalChessPT.x > -1
            && pDlg->m_ChessBoard.m_LocalChessPT.y > -1)
        {
        //发送数据报
        pDlg->m_ChessBoard.m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        }
        else                                                //用户还没有开始下棋
        {
            MessageBox("当前不允许悔棋!","提示");
        }
    }
}
```

（2）对方接收到悔棋消息，判断是否同意悔棋，如果同意，则进行悔棋处理，并向对方发送同意悔棋消息；如果不同意悔棋，则发送拒绝悔棋消息。

```
else if (tcpPackage.cmdType == CT_BACKREQUEST)              //对方发送悔棋请求
{
    if (MessageBox("是否同意悔棋?","提示",MB_YESNO)==IDYES)
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //同意悔棋
```

```
tcpPackage.cmdType = CT_BACKACCEPTANCE;
m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
//进行本地的悔棋处理
if (m_IsDown==TRUE)                                     //该本地下棋了，只需要撤销一步
{
    int nPosX = m_RemoteChessPT.x;
    int nPosY = m_RemoteChessPT.y;
    if (nPosX > -1 && nPosY > -1)                       //用户已下棋
    {
        //重新设置棋子颜色
        m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
        NODE *pNode = new NODE();                       //定义一个棋子
        //复制棋子信息
        memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
        m_BackPlayList.AddTail(pNode);                  //向回放列表中添加棋子
        Invalidate();                                   //刷新窗口
    }
    m_IsDown = FALSE;
}
else                                                    //该对方下棋了，需要撤销两步
{
    int nPosX = m_LocalChessPT.x;                       //获取用户最近放置棋子的坐标
    int nPosY = m_LocalChessPT.y;
    if (nPosX > -1 && nPosY > -1)
    {
        //重新设置棋子颜色
        m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
        NODE *pNode = new NODE();                       //定义棋子
        //复制棋子信息
        memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
        m_BackPlayList.AddTail(pNode);                  //向回放列表中添加棋子
    }
    nPosX = m_RemoteChessPT.x;                          //获取对方最近放置的棋子坐标
    nPosY = m_RemoteChessPT.y;
    if (nPosX > -1 && nPosY > -1)
    {
        //重新设置棋子颜色
        m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
        NODE *pNode = new NODE();                       //定义棋子
        //复制棋子信息
        memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
        m_BackPlayList.AddTail(pNode);                  //向回放列表中添加棋子
    }
    Invalidate();                                       //刷新窗口
}
m_LocalChessPT.x = -1;
m_LocalChessPT.y = -1;
m_RemoteChessPT.x = -1;
m_RemoteChessPT.y = -1;
```

```
        }
        else                                                //拒绝悔棋
        {
            tcpPackage.cmdType = CT_BACKDENY;               //设置数据包命令类型表示拒绝悔棋
            //发送悔棋数据报
            m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        }
}
```

（3）发送请求的一方收到对方的答复信息，决定同意悔棋还是拒绝悔棋。如果同意悔棋，则进行悔棋操作；如果对方拒绝了悔棋，则弹出消息提示框。

```
else if (tcpPackage.cmdType == CT_BACKDENY)             //对方拒绝悔棋
{
        MessageBox("对方拒绝悔棋!","提示");
}
else if (tcpPackage.cmdType == CT_BACKACCEPTANCE)       //对方同意悔棋
{
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //判断是否该本地用户下棋了，如果是则需要撤销之前对方下的棋子，然后再撤销本地用户下的棋子
        if (pDlg->m_ChessBoard.m_IsDown==TRUE)
        {
            int nPosX = m_RemoteChessPT.x;              //获取对方最近放置的棋子坐标
            int nPosY = m_RemoteChessPT.y;
            if (nPosX > -1 && nPosY > -1)
            {
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN; //重新设置棋子颜色
                NODE *pNode = new NODE();               //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);          //将棋子添加到回放列表中
            }
            nPosX = m_LocalChessPT.x;                   //获取用户最近放置的棋子坐标
            nPosY = m_LocalChessPT.y;
            if (nPosX > -1 && nPosY > -1)
            {
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN; //重新设置棋子颜色
                NODE *pNode = new NODE();               //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);          //将棋子添加到回放列表中
            }
            Invalidate();                               //刷新窗口

        }
        else                                            //该对方下棋了，只撤销本地用户下的棋子
        {
            int nPosX = m_LocalChessPT.x;               //获取用户最近放置的棋子坐标
            int nPosY = m_LocalChessPT.y;
```

```
            if (nPosX > -1 && nPosY > -1)
            {
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;  //重新设置棋子颜色
                NODE *pNode = new NODE();                     //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);                //将棋子添加到回放列表中
                Invalidate();                                 //刷新窗口
                m_IsDown = TRUE;
            }
        }
        m_LocalChessPT.x = -1;
        m_LocalChessPT.y = -1;
        m_RemoteChessPT.x = -1;
        m_RemoteChessPT.y = -1;
}
```

10.4.7　设计游戏回放功能

为了让游戏的双方了解下棋的整个过程，在网络五子棋模块中设计了游戏回放功能。当游戏结束时，用户可以通过游戏回放了解整个下棋的过程，分析对方下棋的思路，总结经验。

为了实现游戏回放功能，需要在用户或对方放置棋子时使用链表记录棋子，在回放时遍历链表，输出每一个棋子。思路虽然简单，实现起来却不容易。尤其是在用户悔棋时，需要从棋盘中撤销棋子，如何在链表中记录？此处采用的方法是在用户悔棋时依然向链表中添加悔棋的棋子，只是棋子的颜色为 ncUNKOWN。在游戏回放时，如果链表中的棋子颜色为 ncUNKOWN，将使用背景位图覆盖当前棋子，这样就演示了用户的悔棋效果。下面以代码的形式描述游戏回放功能的实现。

（1）定义游戏回放的链表对象。

```
CPtrList m_BackPlayList;                              //记录用户下棋的步骤
```

（2）在用户放置棋子时向链表中添加棋子。

```
NODE *pNode = new NODE();                             //定义棋子
memcpy(pNode,node,sizeof(NODE));                      //复制棋子信息
m_BackPlayList.AddTail(pNode);                        //将棋子添加到回放列表中
```

（3）在对方放置棋子时在棋盘中显示棋子，向链表中添加棋子，记录对方放置的棋子坐标。

```
else if (tcpPackage.cmdType == CT_POINT)              //客户端棋子坐标信息
{
    int nX = tcpPackage.chessPT.x;                    //获取棋子坐标
    int nY = tcpPackage.chessPT.y;
    m_NodeList[nX][nY].m_Color = ncWHITE;             //设置棋子颜色
    NODE *pNode = new NODE();                         //定义棋子
    memcpy(pNode,&m_NodeList[nX][nY],sizeof(NODE));   //复制棋子信息
    m_BackPlayList.AddTail(pNode);                    //将棋子添加到回放列表中
```

```
        m_RemoteChessPT.x = nX;                              //记录对方放置的棋子坐标
        m_RemoteChessPT.y = nY;
        OnPaint();                                           //重新绘制窗口，显示棋子
        m_IsDown = TRUE;                                     //轮到用户下棋
}
```

（4）在游戏回放时遍历链表，将链表中的每个棋子绘制在棋盘中。如果棋子的颜色为 ncUNKOWN，表示用户进行了悔棋操作，将使用背景位图填充原来的棋子区域，这将导致棋盘中当前棋子的部分表格被填充。因此，在绘制完背景位图之后，还需要绘制部分表格。

```
//游戏回放
void CChessBorad::GamePlayBack()
{
        CDC* pDC = GetDC();                                  //获取窗口设备上下文
        CDC memDC;                                           //定义内存设备上下文
        CBitmap BmpWhite,BmpBlack,BmpBK;                     //定义棋子位图
        memDC.CreateCompatibleDC(pDC);                       //创建内存设备上下文
        BmpBlack.LoadBitmap(IDB_BLACK);                      //加载棋子位图
        BmpWhite.LoadBitmap(IDB_WHITE);
        BmpBK.LoadBitmap(IDB_BLANK);
        BITMAP bmpInfo;                                      //定义位图信息对象
        BmpBlack.GetBitmap(&bmpInfo);                        //获取位图信息
        int nBmpWidth = bmpInfo.bmWidth;                     //获取位图宽度和高度
        int nBmpHeight = bmpInfo.bmHeight;
        POSITION pos = NULL;
        m_bBackPlay = FALSE;
        InitBackPlayNode();                                  //初始化回放列表
        OnPaint();                                           //刷新窗口
        m_bBackPlay = TRUE;
        for(pos = m_BackPlayList.GetHeadPosition(); pos != NULL;)   //遍历回放列表
        {
            NODE* pNode = (NODE*)m_BackPlayList.GetNext(pos);  //获取棋子
            int nPosX,nPosY;
            nPosX = 10*m_fRateX;
            nPosY = 10*m_fRateY;
            pNode->m_IsUsed = TRUE;                          //棋子被使用
            if (pNode->m_Color == ncWHITE)                   //如果为白色棋子
            {
                memDC.SelectObject(&BmpWhite);               //选中白色位图
                //绘制白色棋子
                pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                    nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
            }
            else if (pNode->m_Color == ncBLACK)              //如果为黑色棋子
            {
                memDC.SelectObject(&BmpBlack);               //选中黑色位图
                //绘制黑色棋子
                pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
```

```
                nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
        }
        else if (pNode->m_Color == ncUNKOWN)                    //棋子颜色位置
        {
            memDC.SelectObject(&BmpBK);                         //选中背景颜色
            //绘制背景颜色取消原来显示的棋子
            pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
            //绘制棋盘的局部表格
            //首先获取中心点坐标
            int nCenterX = pNode->m_Point.x ;                   //获取棋子坐标
            int nCenterY = pNode->m_Point.y;
            CPoint topPT(nCenterX,nCenterY-nPosY);
            CPoint bottomPT(nCenterX,nCenterY+nPosY + 5);
            CPen pen(PS_SOLID,1,RGB(0,0,0));                    //定义黑色画笔
            pDC->SelectObject(&pen);                            //选中画笔
            pDC->MoveTo(topPT);                                 //绘制直线
            pDC->LineTo(bottomPT);
            CPoint leftPT(nCenterX-nPosX,nCenterY);
            CPoint rightPT(nCenterX+nPosX + 10 ,nCenterY);
            pDC->MoveTo(leftPT);                                //绘制横线
            pDC->LineTo(rightPT);
        }
        //延迟
        SYSTEMTIME beginTime,endTime;
        GetSystemTime(&beginTime);
        if (beginTime.wSecond > 58)
            beginTime.wSecond = 58;
        while (true)                                            //进行延迟操作
        {
            MSG msg;                                            //在回放过程中相应的界面操作
            ::GetMessage(&msg,0,0,WM_USER);
            TranslateMessage(&msg);
            DispatchMessage(&msg);
            GetSystemTime(&endTime);
            if (endTime.wSecond ==0 )
                endTime.wSecond = 59;
            if (endTime.wSecond > beginTime.wSecond)
                break;
        }
    }
    BmpWhite.DeleteObject();                                    //释放位图对象
    BmpBlack.DeleteObject();
    BmpBK.DeleteObject();
    memDC.DeleteDC();                                           //释放内存设备上下文
    MessageBox("游戏回放结束!","提示");
}
```

（5）在窗口需要绘制时（WM_PAINT 消息处理函数中），如果当前处于回放状态，则保持回放的效果。

```
if (m_bBackPlay)                                                    //当前是否为游戏回放
{
    POSITION pos = NULL;
    for(pos = m_BackPlayList.GetHeadPosition(); pos != NULL;)       //遍历回放链表
    {
        NODE* pNode = (NODE*)m_BackPlayList.GetNext(pos);           //获取节点
        if (pNode->m_IsUsed==TRUE)                                  //判断节点是否被使用
        {
            int nPosX,nPosY;
            nPosX = 10*m_fRateX;
            nPosY = 10*m_fRateY;
            if (pNode->m_Color == ncWHITE)                          //如果为白色棋子
            {
                memDC.SelectObject(&BmpWhite);                      //选中白色棋子位图
                //绘制白色棋子
                pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                    nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
            }
            else if (pNode->m_Color == ncBLACK)                     //如果为黑色棋子
            {
                memDC.SelectObject(&BmpBlack);                      //选中黑色棋子位图
                //绘制黑色棋子
                pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                    nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
            }
            else if (pNode->m_Color == ncUNKOWN)                    //棋子颜色未知
            {
                memDC.SelectObject(&BmpBK);                         //选中背景位图
                //绘制背景位图
                pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                    nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
                //绘制棋盘的局部表格，首先获取中心点坐标
                int nCenterX = pNode->m_Point.x ;
                int nCenterY = pNode->m_Point.y;
                CPoint topPT(nCenterX,nCenterY-nPosY);
                CPoint bottomPT(nCenterX,nCenterY+nPosY + 5);
                CPen pen(PS_SOLID,1,RGB(0,0,0));                    //定义黑色画笔
                pDC->SelectObject(&pen);                            //选中画笔
                pDC->MoveTo(topPT);                                 //绘制直线
                pDC->LineTo(bottomPT);
                CPoint leftPT(nCenterX-nPosX,nCenterY);
                CPoint rightPT(nCenterX+nPosX + 10 ,nCenterY);
                pDC->MoveTo(leftPT);                                //绘制横线
```

```
                    pDC->LineTo(rightPT);
                }
            }
        }
}
```

10.4.8　对方网络状态测试

在进行游戏的过程中，为了防止由于网络故障或某一方掉线致使游戏无法结束或无法重新开始，于是在网络五子棋模块中添加了网络状态测试功能。实现网络状态测试功能比较简单，在游戏开始后，由服务器方每隔 1 秒向对方发送网络状态测试信息，然后开启一个计时器，对方在接收到网络状态测试信息后发送应答信息到服务器。在服务器端，如果 3 秒后没有收到应答信息，则表示与对方失去连接，当前游戏结束。在客户端，如果 3 秒后没有收到服务器端发来的网络状态测试信息，则认为与对方失去连接，当前游戏结束。关键代码如下。

（1）在游戏过程中，服务器端每隔 1 秒发送网络状态测试信息。

```
void CChessBorad::OnTimer(UINT nIDEvent)
{
    if (m_IsConnect)                                            //客户已连接服务器
    {
        TCP_PACKAGE tcpPackage;                                 //定义数据包
        tcpPackage.cmdType = CT_NETTEST;                        //设置数据包类型
        m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));     //发送网络测试信息
        m_TestNum++;                                            //累加计数
        if (m_TestNum > 3)                                      //对方掉线，游戏结束
        {
            m_TestNum = 0;
            m_IsDown = FALSE;
            m_IsStart = FALSE;
            m_IsWin = FALSE;
            m_State = esEND;
            m_IsConnect = FALSE;
            InitializeNode();                                   //初始化棋子
            //获取父窗口
            CSrvFiveChessDlg * pDlg = (CSrvFiveChessDlg*)GetParent();
            pDlg->m_RightPanel.m_NetState.SetWindowText("网路状态:断开连接");
            Invalidate();                                       //更新界面
            //初始化最近放置的棋子坐标
            m_LocalChessPT.x = m_LocalChessPT.y = -1;
            m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        }
    }
    CDialog::OnTimer(nIDEvent);
}
```



（2）游戏开始时，在客户端开启一个定时器，检测是否收到服务器端的网络状态测试信息。

```
void CChessBorad::OnTimer(UINT nIDEvent)
{
    if (m_IsConnect)
    {
        m_TestNum++;                                          //累加计数
        if (m_TestNum > 3)                                    //与对方断开连接
        {
            m_TestNum = 0;
            m_IsConnect = FALSE;
            m_IsDown = FALSE;
            m_IsStart = FALSE;
            m_IsWin = FALSE;
            m_State = esEND;
            m_IsConnect = FALSE;
            InitializeNode();                                 //初始化所有棋子
            CClientFiveChessDlg * pDlg = (CClientFiveChessDlg*)GetParent();
            pDlg->m_RightPanel.m_NetState.SetWindowText("网路状态:断开连接");
            Invalidate();                                     //更新界面
            m_LocalChessPT.x = m_LocalChessPT.y = -1;         //初始化最近放置的棋子坐标
            m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        }
    }
    CDialog::OnTimer(nIDEvent);
}
```

（3）在客户端，如果收到服务器端的网络状态测试信息，则将计数器归零。

```
TCP_PACKAGE tcpPackage;                                       //定义数据包
memcpy(&tcpPackage,pBuffer,sizeof(TCP_PACKAGE));              //复制数据包
if (tcpPackage.cmdType == CT_NETTEST)                         //测试网络状态
{
    m_TestNum = 0;                                            //接收到网络状态测试，计数为 0
    //向服务器发送网络状态测试信息
    m_ClientSoc k.Send(&tcpPackage,sizeof(TCP_PACKAGE));
}
```

10.5　服务器端主窗体设计


视频讲解

10.5.1　服务器端主窗体概述

服务器端的主窗体主要由游戏控制窗体、棋盘窗体和对方信息窗体 3 个子窗体构成，运行效果如图 10.10 所示。

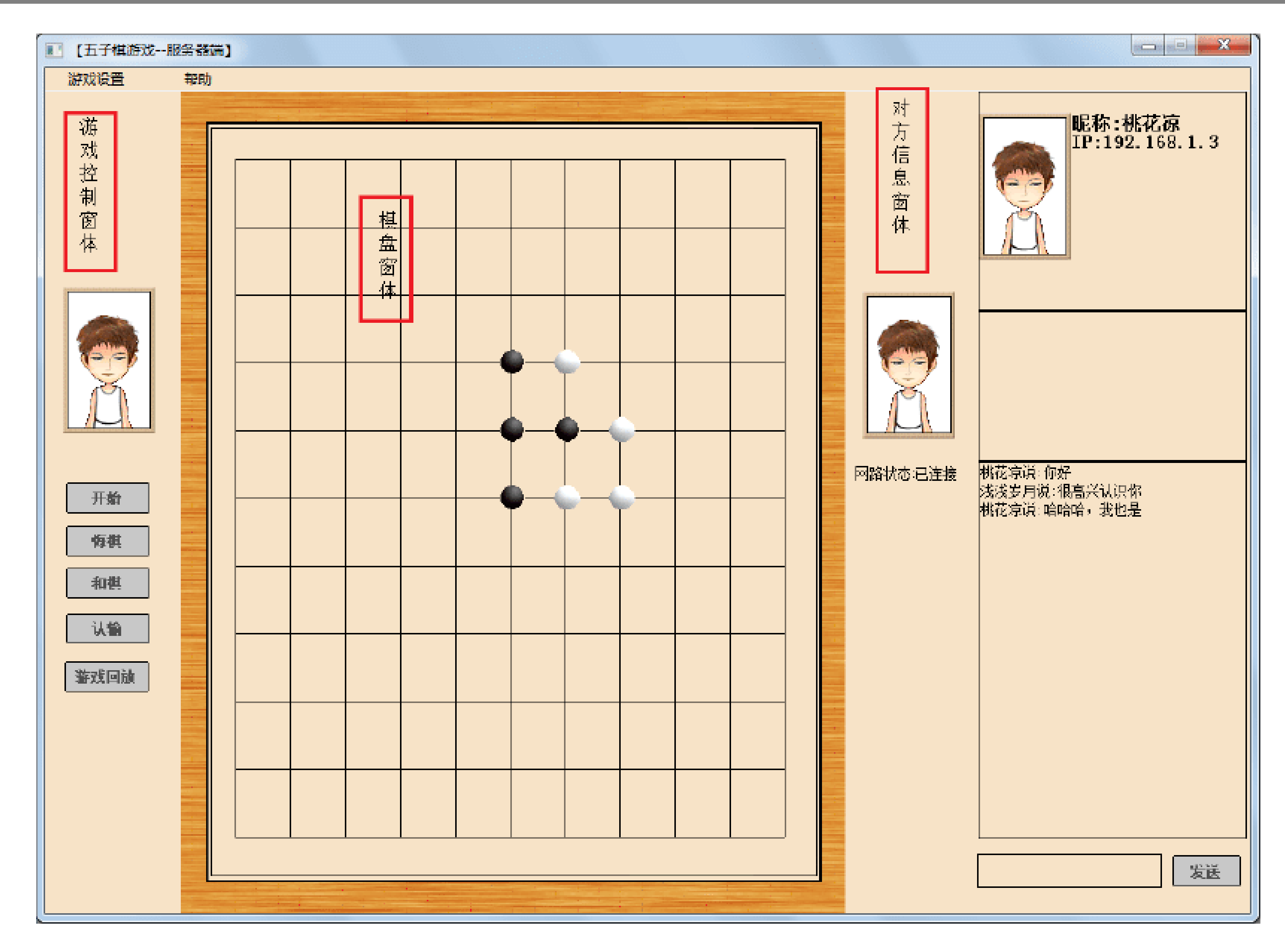


图 10.10　网络五子棋服务器端主窗体的运行效果

10.5.2　服务器端主窗体实现过程

服务器端主窗体实现过程如下：

（1）创建一个基于窗口的工程，工程名称为 SrvFiveChess。工程向导将创建一个默认的对话框类——CSrvFiveChessDlg，该类将作为网络五子棋服务器端的主窗体。

（2）定义 3 个子窗体变量，分别表示游戏控制窗体、棋盘窗体和对方信息窗体（有关这 3 个窗体的设计过程将在后面几节中进行介绍）。

```
CLeftPanel m_LeftPanel;                    //游戏控制窗体
CRightPanel m_RightPanel;                  //对方信息窗体
CChessBorad m_ChessBoard;                  //棋盘窗体
```

（3）在窗口初始化时创建游戏控制窗体、棋盘窗体和对方信息窗体，并调整这 3 个窗体的大小和位置。

```
BOOL CSrvFiveChessDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
```

```
{
	CString strAboutMenu;
	strAboutMenu.LoadString(IDS_ABOUTBOX);
	if (!strAboutMenu.IsEmpty())
	{
		pSysMenu->AppendMenu(MF_SEPARATOR);
		pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
	}
}

SetIcon(m_hIcon, TRUE);                    //设置大图标
SetIcon(m_hIcon, FALSE);                   //设置小图标

HMENU hMenu = GetMenu()->GetSafeHmenu();
if (hMenu != NULL)
{
	m_Menu.AttatchMenu(hMenu);
	m_Menu.SetMenuItemInfo(&m_Menu);
}

AfxInitRichEdit();
//创建窗口列表
m_RightPanel.Create(IDD_RIGHTPANEL_DIALOG,this);
m_RightPanel.ShowWindow(SW_SHOW);
CRect wndRC;
m_RightPanel.GetWindowRect(wndRC);
int nWidth = wndRC.Width();

CRect cltRC;                               //客户区域
GetClientRect(cltRC);
int nHeight = cltRC.Height();

//定义窗口列表显示的区域
CRect pnlRC;
pnlRC.left = cltRC.right-nWidth;
pnlRC.top = 0;
pnlRC.bottom = nHeight;
pnlRC.right = cltRC.right;
//显示窗口
m_RightPanel.MoveWindow(pnlRC);

//记录右边窗口的宽度
int nRightWidth = nWidth;
//创建左边的列表信息窗口
m_LeftPanel.Create(IDD_LEFTPANEL_DIALOG,this);
m_LeftPanel.ShowWindow(SW_SHOW);

m_LeftPanel.GetWindowRect(wndRC);
nWidth = wndRC.Width();
```

```
        pnlRC.left = 0;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
        pnlRC.right = nWidth;

        //记录左边窗口的宽度
        int nLeftWidth = nWidth;
        //显示窗口
        m_LeftPanel.MoveWindow(pnlRC);

        //创建棋盘窗口
        m_ChessBoard.Create(IDD_CHESSBORAD_DIALOG,this);
        m_ChessBoard.ShowWindow(SW_SHOW);
        //计算棋盘的显示区域
        pnlRC.left = nLeftWidth;                              //为左边窗口的宽度
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;                               //主窗口的高度
        pnlRC.right = cltRC.Width() - nRightWidth;            //整个窗口的区域去除右边窗口的宽度
        m_ChessBoard.MoveWindow(pnlRC);
        m_bCreatePanel = TRUE;
        return TRUE;
}
BOOL CSrvFiveChessDlg::OnInitDialog()
{
        …//省略不必要的代码
        m_RightPanel.Create(IDD_RIGHTPANEL_DIALOG,this);      //创建对方信息窗体
        m_RightPanel.ShowWindow(SW_SHOW);                     //显示对方信息窗体
        CRect wndRC;
        m_RightPanel.GetWindowRect(wndRC);                    //获取窗体区域
        int nWidth = wndRC.Width();                           //获取窗体宽度
        CRect cltRC;
        GetClientRect(cltRC);                                 //获取主窗体客户区域
        int nHeight = cltRC.Height();                         //获取主窗体高度
        //定义对方信息窗口显示的区域
        CRect pnlRC;
        pnlRC.left = cltRC.right-nWidth;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
        pnlRC.right = cltRC.right;
        m_RightPanel.MoveWindow(pnlRC);                       //设置对方信息窗体显示区域
        int nRightWidth = nWidth;                             //记录右边窗体的宽度
        m_LeftPanel.Create(IDD_LEFTPANEL_DIALOG,this);        //创建游戏控制窗体
        m_LeftPanel.ShowWindow(SW_SHOW);                      //显示游戏控制窗体
        m_LeftPanel.GetWindowRect(wndRC);                     //获取游戏控制窗体区域
        nWidth = wndRC.Width();                               //获取窗体宽度
        pnlRC.left = 0;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
```



```
    pnlRC.right = nWidth;
    int nLeftWidth = nWidth;                              //记录游戏控制窗体的宽度
    m_LeftPanel.MoveWindow(pnlRC);                        //显示游戏控制窗体
    m_ChessBoard.Create(IDD_CHESSBORAD_DIALOG,this);      //创建棋盘窗体
    m_ChessBoard.ShowWindow(SW_SHOW);                     //显示棋盘窗体
    //计算棋盘的显示区域
    pnlRC.left = nLeftWidth;                              //为游戏控制窗体的宽度
    pnlRC.top = 0;
    pnlRC.bottom = nHeight;                               //主窗体的高度
    pnlRC.right = cltRC.Width() - nRightWidth;            //整个窗体的区域去除对方信息窗体的宽度
    m_ChessBoard.MoveWindow(pnlRC);                       //设置棋盘窗体显示区域
    m_bCreatePanel = TRUE;
    return TRUE;
}
```

（4）在窗口大小改变时，调整游戏控制窗体、棋盘窗体和对方信息窗体的大小和位置。

```
void CSrvFiveChessDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
    if (m_bCreatePanel)                                   //判断子窗体是否被创建
    {
        CRect wndRC;
        m_RightPanel.GetWindowRect(wndRC);                //获取对方信息窗体区域
        int nWidth = wndRC.Width();                       //获取对方信息窗体宽度
        CRect cltRC;
        GetClientRect(cltRC);                             //获取主窗体客户区域
        int nHeight = cltRC.Height();                     //获取主窗体高度
        //定义窗口列表显示的区域
        CRect pnlRC;
        pnlRC.left = cltRC.right-nWidth;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
        pnlRC.right = cltRC.right;
        m_RightPanel.MoveWindow(pnlRC);                   //设置对方信息窗体显示区域
        int nRightWidth = nWidth;                         //获取对方信息窗体宽度
        m_RightPanel.Invalidate();                        //更新对方信息窗体
        //显示左边的窗口列表区域
        m_LeftPanel.GetWindowRect(wndRC);
        nWidth = wndRC.Width();
        pnlRC.left = 0;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
        pnlRC.right = nWidth;
        m_LeftPanel.MoveWindow(pnlRC);                    //设置游戏控制窗体显示区域
        int nLeftWidth = nWidth;                          //获取游戏控制窗体宽度
        pnlRC.left = nLeftWidth;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;                           //获取主窗体的高度
```

```
            //整个窗体的区域去除右边窗体的宽度
            pnlRC.right = cltRC.Width() - nRightWidth;
            m_ChessBoard.MoveWindow(pnlRC);                    //设置棋盘窗体显示区域
            m_ChessBoard.Invalidate();                         //更新棋盘窗体
        }
}
```

（5）处理窗口的 WM_GETMINMAXINFO 消息，限制窗口的最小窗体大小。

```
void CSrvFiveChessDlg::OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI)
{
        lpMMI->ptMinTrackSize.x = 800;                         //限制窗体最小宽度
        lpMMI->ptMinTrackSize.y = 500;                         //限制窗体最小高度
        CDialog::OnGetMinMaxInfo(lpMMI);
}
```



10.6　棋盘窗体模块设计

10.6.1　棋盘窗体模块概述

棋盘窗体是整个网络五子棋模块的核心。在棋盘窗体中实现的主要功能包括接受客户端连接、接收客户端发送的数据、绘制棋盘、绘制棋盘表格、绘制棋子、赢棋判断、网络状态测试、开始游戏、游戏回放等。棋盘窗体的运行效果如图 10.11 所示。

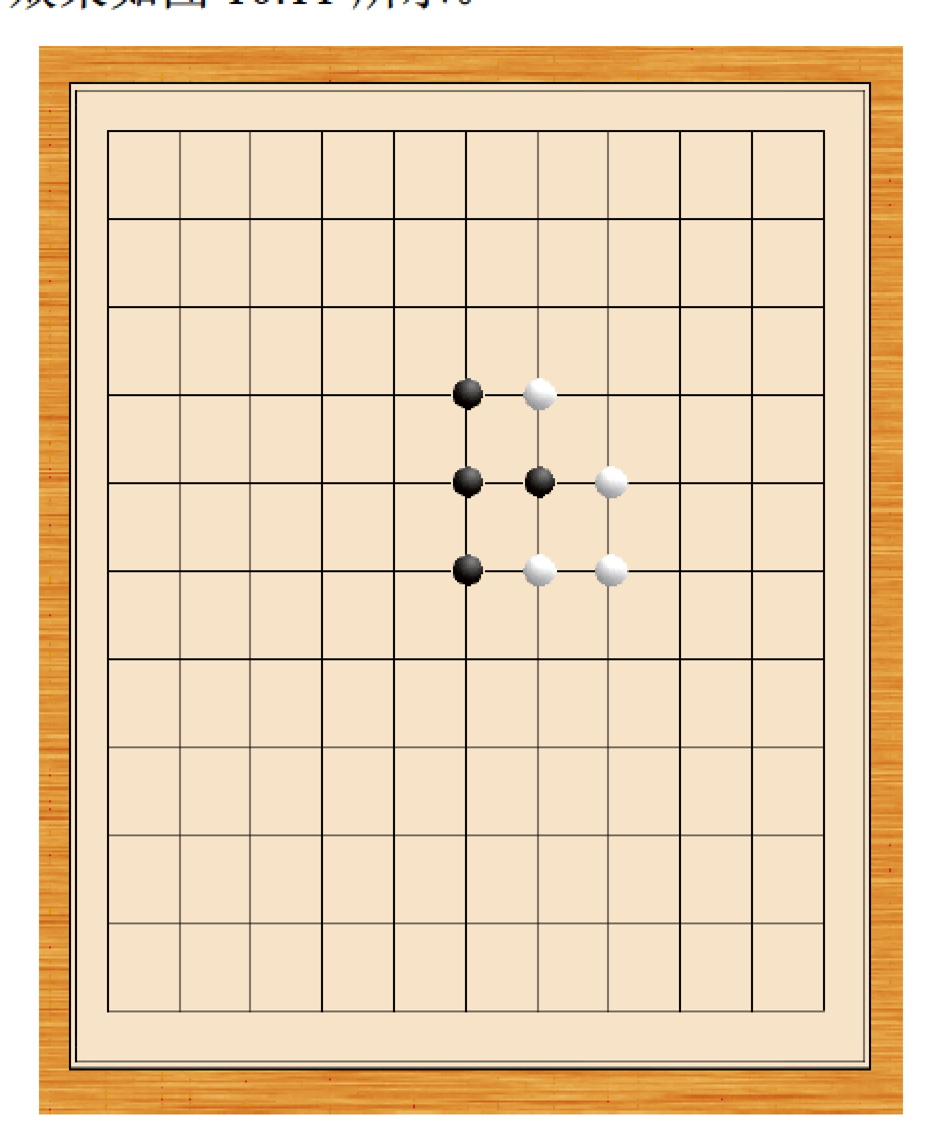


图 10.11　棋盘窗体的运行效果





10.6.2　棋盘窗体模块界面布局

棋盘窗体界面布局如下：

（1）创建一个对话框类，类名为 CChessBorad。

（2）设置对话框属性，如表 10.1 所示。

表 10.1　棋盘窗体属性设置

控件 ID	控 件 属 性	关 联 变 量
IDD_CHESSBORAD_DIALOG	Style：Child Border：None Title bar：FALSE	CChessBorad：m_ChessBoard

10.6.3　棋盘窗体模块实现过程

棋盘窗体模块实现过程如下：

（1）向对话框类中添加 AcceptConnection 方法，当客户端有套接字连接时，接受客户端的连接，记录客户端的 IP、端口号和连接时间。

```
void CChessBorad::AcceptConnection()
{
	m_ClientSock.Close();                                          //关闭客户端套接字
	m_SrvSock.Accept(m_ClientSock);                                //接受客户端连接
	m_IsConnect = TRUE;
	CSrvFiveChessDlg * pDlg = (CSrvFiveChessDlg*)GetParent();      //获取主窗口
	CTime tmNow = CTime::GetCurrentTime();                         //获取当前时间
	CString csFormat = tmNow.Format("%H:%M:%S");
	pDlg->m_RightPanel.m_UserList.SetItemText(0,2,csFormat);
	CString csClientIP;
	UINT nPort;
	m_ClientSock.GetSockName(csClientIP,nPort);                    //获取客户端信息
	if (pDlg->m_RightPanel.m_UserList.GetItemCount()<2)
	{
		//向用户列表中添加数据
		int nItem = pDlg->m_RightPanel.m_UserList.InsertItem(1,"");
		pDlg->m_RightPanel.m_UserList.SetItemText(nItem,1,csClientIP);
		pDlg->m_RightPanel.m_UserList.SetItemText(nItem,2,csFormat);
	}
	else
	{
		//设置用于列表中的数据
		pDlg->m_RightPanel.m_UserList.SetItemText(1,1,csClientIP);
		pDlg->m_RightPanel.m_UserList.SetItemText(1,2,csFormat);
	}
```

```
    pDlg->m_RightPanel.m_NetState.SetWindowText("网路状态:已连接");
}
```

（2）向对话框类中添加 DrawChessboard 方法，在棋盘位图背景上绘制表格。

```
void CChessBorad::DrawChessboard()
{
    CDC* pDC = GetDC();                                         //获取窗口设备上下文
    CPen pen(PS_SOLID,1,RGB(0,0,0));                            //定义黑色的画笔
    pDC->SelectObject(&pen);                                    //选中画笔
    int nOriginX = m_nOrginX*m_fRateX;                          //计算表格的起点坐标
    int nOriginY = m_nOrginY*m_fRateY;
    int nCellWidth = m_nCellWidth*m_fRateX;                     //计算单元格的宽度和高度
    int nCellHeight = m_nCellHeight*m_fRateY;
    for (int i = 0; i<m_nRowCount+1; i++)                       //绘制棋盘中的列
    {
        pDC->MoveTo(nOriginX+nCellWidth*(i),nOriginY);
        pDC->LineTo(nOriginX+nCellWidth*(i),nOriginY+m_nRowCount*nCellHeight);
    }
    for (int j = 0; j<m_nColCount+1; j++)                       //绘制棋盘中的行
    {
        pDC->MoveTo(nOriginX ,nOriginY+(j)*nCellHeight);
        pDC->LineTo(nOriginX +m_nColCount*nCellWidth,nOriginY+(j)*nCellHeight);
    }
}
```

（3）向对话框类中添加 FreeBackPlayList 方法，释放游戏回放使用的链表。

```
void CChessBorad::FreeBackPlayList()
{
    if (m_BackPlayList.GetCount()>0)                            //判断回放列表中是否有棋子
    {
        POSITION pos;
        //遍历回放列表
        for (pos = m_BackPlayList.GetHeadPosition();pos != NULL;)
        {
            NODE *pNode = (NODE*)m_BackPlayList.GetNext(pos);   //获取棋子
            delete pNode;                                       //释放棋子
        }
        m_BackPlayList.RemoveAll();                             //移除所有棋子
    }
}
```

（4）向对话框类中添加 GamePlayBack 方法，实现游戏回放。在游戏回放时遍历链表，将链表中的每个棋子绘制在棋盘中。如果棋子的颜色为 ncUNKOWN，表示用户进行了悔棋操作，将使用背景位图填充原来的棋子区域，这将导致棋盘中当前棋子的部分表格被填充。因此，在绘制完背景位图之后，还需要绘制部分表格。



```
void CChessBorad::GamePlayBack()
{
    CDC* pDC = GetDC();                                    //获取窗口设备上下文
    CDC memDC;                                             //定义内存设备上下文
    CBitmap BmpWhite,BmpBlack,BmpBK;                       //定义棋子位图
    memDC.CreateCompatibleDC(pDC);                         //创建内存设备上下文
    BmpBlack.LoadBitmap(IDB_BLACK);                        //加载棋子位图
    BmpWhite.LoadBitmap(IDB_WHITE);
    BmpBK.LoadBitmap(IDB_BLANK);
    BITMAP bmpInfo;                                        //定义位图信息对象
    BmpBlack.GetBitmap(&bmpInfo);                          //获取位图信息
    int nBmpWidth = bmpInfo.bmWidth;                       //获取位图宽度和高度
    int nBmpHeight = bmpInfo.bmHeight;
    POSITION pos = NULL;
    m_bBackPlay = FALSE;
    InitBackPlayNode();                                    //初始化回放列表
    OnPaint();                                             //刷新窗口
    m_bBackPlay = TRUE;
    for(pos = m_BackPlayList.GetHeadPosition(); pos != NULL;)  //遍历回放列表
    {
        NODE* pNode = (NODE*)m_BackPlayList.GetNext(pos);  //获取棋子
        int nPosX,nPosY;
        nPosX = 10*m_fRateX;
        nPosY = 10*m_fRateY;
        pNode->m_IsUsed = TRUE;                            //棋子被使用
        if (pNode->m_Color == ncWHITE)                     //如果为白色棋子
        {
            memDC.SelectObject(&BmpWhite);                 //选中白色位图
            //绘制白色棋子
            pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
        }
        else if (pNode->m_Color == ncBLACK)                //如果为黑色棋子
        {
            memDC.SelectObject(&BmpBlack);                 //选中黑色位图
            //绘制黑色棋子
            pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
        }
        else if (pNode->m_Color == ncUNKOWN)               //棋子颜色位置
        {
            memDC.SelectObject(&BmpBK);                    //选中背景颜色
            //绘制背景颜色取消原来显示的棋子
            pDC->StretchBlt(pNode->m_Point.x-nPosX,pNode->m_Point.y-nPosY,
                nBmpWidth,nBmpHeight,&memDC,0,0,nBmpWidth,nBmpHeight,SRCCOPY);
            //绘制棋盘的局部表格
            //首先获取中心点坐标
            int nCenterX = pNode->m_Point.x ;              //获取棋子坐标
            int nCenterY = pNode->m_Point.y;
```

```
                CPoint topPT(nCenterX,nCenterY-nPosY);
                CPoint bottomPT(nCenterX,nCenterY+nPosY + 5);
                CPen pen(PS_SOLID,1,RGB(0,0,0));                    //定义黑色画笔
                pDC->SelectObject(&pen);                            //选中画笔
                pDC->MoveTo(topPT);                                 //绘制直线
                pDC->LineTo(bottomPT);
                CPoint leftPT(nCenterX-nPosX,nCenterY);
                CPoint rightPT(nCenterX+nPosX + 10 ,nCenterY);
                pDC->MoveTo(leftPT);                                //绘制横线
                pDC->LineTo(rightPT);
            }
            //延迟
            SYSTEMTIME beginTime,endTime;
            GetSystemTime(&beginTime);
            if (beginTime.wSecond > 58)
                beginTime.wSecond = 58;
            while (true)                                            //进行延迟操作
            {
                MSG msg;                                            //在回放过程中相应界面操作
                ::GetMessage(&msg,0,0,WM_USER);
                TranslateMessage(&msg);
                DispatchMessage(&msg);
                GetSystemTime(&endTime);
                if (endTime.wSecond ==0)
                    endTime.wSecond = 59;
                if (endTime.wSecond > beginTime.wSecond)
                    break;
            }
        }
        BmpWhite.DeleteObject();                                    //释放位图对象
        BmpBlack.DeleteObject();
        BmpBK.DeleteObject();
        memDC.DeleteDC();                                           //释放内存设备上下文
        MessageBox("游戏回放结束!","提示");
}
```

（5）向对话框类中添加 GetLikeNode 方法，根据坐标点获取相应的棋子。

```
NODE* CChessBorad::GetLikeNode(CPoint pt)
{
        CPoint tmp;
        for (int i = 0 ;i<m_nRowCount+1;i++)                        //遍历行
            for (int j = 0; j<m_nColCount+1;j++)                    //遍历列
            {
                tmp = m_NodeList[i][j].m_Point;                     //获取棋子坐标
                int nSizeX = 10 * m_fRateX;
                int nSizeY = 10 * m_fRateY;
                //定义一个临近棋子的区域
                CRect rect(tmp.x-nSizeX,tmp.y-nSizeY,tmp.x+nSizeX,tmp.y+nSizeY);
                if (rect.PtInRect(pt))                              //判断鼠标指针是否在临近区域
```

```
                return &m_NodeList[i][j];                       //返回棋子
        }
    return NULL;
}
```

（6）向对话框类中添加 GetNodeFromPoint 方法，根据坐标点获取相应的棋子。与 GetLikeNode 方法不同的是，GetNodeFromPoint 方法进行精确的坐标比较。

```
NODE* CChessBorad::GetNodeFromPoint(CPoint pt)
{
    for (int i=0; i<m_nRowCount+1; i++)                         //遍历行
    {
        for (int j=0; j<m_nColCount+1; j++)                     //遍历列
        {
            if (m_NodeList[i][j].m_Point == pt)                 //匹配棋子坐标
                return &m_NodeList[i][j];                       //返回棋子
        }
    }
    return NULL;
}
```

（7）向对话框类中添加 InitBackPlayNode 方法，初始化游戏回放使用的链表节点。

```
void CChessBorad::InitBackPlayNode()
{
    POSITION pos;
    for(pos = m_BackPlayList.GetHeadPosition(); pos != NULL;)   //遍历回放列表
    {
        NODE* pNode = (NODE*)m_BackPlayList.GetNext(pos);       //获取棋子
        //将棋子设置为未使用状态
        pNode->m_IsUsed = FALSE;
    }
}
```

（8）向对话框类中添加 InitializeNode 方法，初始化棋盘中的所有棋子，将其设置为未使用状态。

```
void CChessBorad::InitializeNode()
{
    for (int i=0; i<m_nRowCount+1; i++)                         //遍历行
    {
        for (int j=0; j<m_nColCount+1; j++)                     //遍历列
        {
            m_NodeList[i][j].m_Color = ncUNKOWN;                //设置棋子颜色
            m_NodeList[i][j].m_nX = i;                          //设置棋子坐标
            m_NodeList[i][j].m_nY = j;
        }
    }
    OnPaint();                                                  //更新窗口
}
```

（9）向对话框类中添加 IsWin 方法，判断是否赢棋。在判断五子棋输赢时需要从水平方向、垂直方向和对角线方向分别进行判断。以从垂直方向判断为例，以当前棋子为中心，向上方查找同一个颜色的棋子，有则累加计数，然后从当前节点的下方开始查找节点，如果有同一个颜色的棋子，继续累加计数，当计数达到 5 时则表示赢棋。

```
NODE* CChessBorad::IsWin(NODE *pCurrent)
{
    if (pCurrent->m_Color != ncBLACK)
        return NULL;
    //按 4 个方向判断
    int num = 0;                                                    //定义计数
    m_Startpt.x = pCurrent->m_nX;
    m_Startpt.y = pCurrent->m_nY;
    m_Endpt.x = pCurrent->m_nX;
    m_Endpt.y = pCurrent->m_nY;
    //按垂直方向判断，在当前节点按上、下两个方向遍历
    NODE* tmp = pCurrent->m_pRecents[1];                            //获得当前节点的上方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)          //遍历上方节点
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                                   //累计连续棋子数量
        if (num >= 4)                                               //是否有 5 个连续棋子
        {
            return tmp;                                             //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[1];
    }
    tmp = pCurrent->m_pRecents[6];                                  //获得当前节点的下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)          //遍历下方节点
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;
        if (num >= 4)
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[6];
    }
    //按水平方向判断，在当前节点，按左、右两个方向遍历
    num = 0;
    tmp = pCurrent->m_pRecents[3];                                  //遍历左节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {

        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
```

```
        num += 1;                                         //累加连续棋子数量
        if (num >= 4)                                     //是否有 5 个连续棋子
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[3];
    }
    tmp = pCurrent->m_pRecents[4];                        //遍历右节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;
        if (num >= 4)
        {
            return tmp;
        }
        tmp = tmp->m_pRecents[4];
    }
    //按 135° 斜角遍历
    num = 0;
    tmp = pCurrent->m_pRecents[0];
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)     //遍历斜上方节点
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                         //累加连续棋子数量
        if (num >= 4)                                     //是否有 5 个连续棋子
        {
            return tmp;                                   //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[0];
    }
    tmp = pCurrent->m_pRecents[7];                        //遍历斜下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;                                         //累加连续棋子数量
        if (num >= 4)                                     //是否有 5 个连续棋子
        {
            return tmp;                                   //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[7];
    }
    //按 45° 斜角遍历
    num = 0;
    tmp = pCurrent->m_pRecents[2];                        //遍历斜上方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
```

```
    {
        m_Startpt.x = tmp->m_nX;
        m_Startpt.y = tmp->m_nY;
        num += 1;                                        //累加连续棋子数量
        if (num >= 4)                                    //是否有 5 个连续棋子
        {
            return tmp;                                  //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[2];
    }
    tmp = pCurrent->m_pRecents[5];                       //遍历斜下方节点
    while (tmp != NULL && tmp->m_Color==pCurrent->m_Color)
    {
        m_Endpt.x = tmp->m_nX;
        m_Endpt.y = tmp->m_nY;
        num += 1;                                        //累加连续棋子数量
        if (num >= 4)                                    //是否有 5 个连续棋子
        {
            return tmp;                                  //表示赢棋，返回最后一个棋子
        }
        tmp = tmp->m_pRecents[5];
    }
    return NULL;
}
```

（10）在对话框初始化时创建套接字，初始化棋子，设置棋子坐标。

```
BOOL CChessBorad::OnInitDialog()
{
    CDialog::OnInitDialog();
    int nOriginX = m_nOrginX*m_fRateX;                   //计算表格起点坐标
    int nOriginY = m_nOrginY*m_fRateY;
    int nCellWidth = m_nCellWidth*m_fRateX;              //计算表格单元格宽度
    int nCellHeight = m_nCellHeight*m_fRateY;
    m_ClientSock.AttachDlg(this);
    m_SrvSock.AttachDlg(this);
    m_ClientSock.Create();                               //创建套接字
    InitializeNode();                                    //初始化棋子
    for (int i=0; i<m_nRowCount+1; i++)
    {
        for (int j=0; j<m_nColCount+1; j++)
        {
            //设置棋子坐标
            m_NodeList[i][j].m_Point= CPoint(nOriginX+nCellWidth*j,nOriginY+nCellHeight*i);
        }
    }
    for (int m=0; m<m_nRowCount+1; m++)                  //为每个棋子设置临近节点
    {
        for (int n=0; n<m_nColCount+1; n++)
```



```
            {
                SetRecentNode(&m_NodeList[m][n]);
            }
        }
        return TRUE;
}
```

（11）在游戏开始时，如果轮到用户下棋，则在当前鼠标附近（与鼠标点最近的棋子坐标点）添加一个棋子，并向对方发送该棋子的坐标。

```
void CChessBorad::OnLButtonUp(UINT nFlags, CPoint point)
{
        CPoint pt = point;
        if (m_IsStart == FALSE)                                 //游戏终止
            return;
        if (m_IsDown==TRUE)                                     //轮到客户端
        {
            NODE* node = GetLikeNode(pt);                       //根据坐标获取棋子
            if (node !=NULL)
            {
                if (node->m_Color==ncUNKOWN)                    //如果棋子被使用
                {
                    node->m_Color = ncBLACK;                    //设置棋子颜色
                    NODE *pNode = new NODE();                   //定义棋子
                    memcpy(pNode,node,sizeof(NODE));            //复制棋子
                    m_BackPlayList.AddTail(pNode);              //将棋子添加到回放列表中
                    OnPaint();                                  //更新窗口
                    TCP_PACKAGE package;                        //定义一个 TCP 数据包
                    package.cmdType = CT_POINT;                 //设置数据包命令类型
                    package.chessPT.x = node->m_nX;             //读取棋子坐标
                    package.chessPT.y = node->m_nY;
                    //将棋子坐标发送到对方
                    m_ClientSock.Send((void*)&package,sizeof(TCP_PACKAGE));
                    m_LocalChessPT.x = node->m_nX;              //记录最近的棋子坐标
                    m_LocalChessPT.y = node->m_nY;
                    m_IsDown= FALSE;
                    if (IsWin(node)!= NULL)                     //进行赢棋判断
                    {
                        m_IsStart = FALSE;
                        Sleep(1000);
                        //赢棋
                        TCP_PACKAGE   winPackage;               //定义数据包
                        winPackage.cmdType = CT_WINPOINT;       //设置数据包命令类型
                        winPackage.winPT[0] = m_Startpt;        //设置赢棋的起点坐标
                        winPackage.winPT[1] = m_Endpt;          //设置赢棋的终点坐标
                        //发送赢棋数据包
                        m_ClientSock.Send((void*)&winPackage,sizeof(TCP_PACKAGE));
                        m_IsWin = TRUE;
                        m_State = esEND;                        //设置游戏结束
```

```
                    Invalidate();                                   //更新窗口
                    MessageBox("恭喜你,赢了!!!");
                    m_IsWin = FALSE;
                    InitializeNode();                               //初始化节点
                    Invalidate();                                   //更新窗口
                    //初始化用户最近放置的棋子坐标
                    m_LocalChessPT.x = m_LocalChessPT.y = -1;
                    m_RemoteChessPT.x = m_RemoteChessPT.y   = -1;
                }
            }
        }
    }
    CDialog::OnLButtonUp(nFlags, point);
}
```

（12）处理“服务器设置”菜单命令的单击事件，设置服务器的地址和端口号，然后将信息添加到对方信息窗体的用户列表中。

```
void CChessBorad::OnMenuSvrSetting()
{
    CServerSetting SrvDlg;                                          //定义服务器设置对话框
    if (m_ConfigSrv == FALSE)                                       //没有配置服务器
    {
        if (SrvDlg.DoModal()==IDOK)
        {
            UINT port = SrvDlg.m_Port;                              //获取端口信息
            //获取主窗口
            CSrvFiveChessDlg * pDlg = (CSrvFiveChessDlg*)GetParent();
            //创建并监听套接字
            if (m_SrvSock.Create(port,SOCK_STREAM,SrvDlg.m_HostIP) && m_SrvSock.Listen())
            {
                m_ConfigSrv = TRUE;                                 //服务器信息已设置
                //向用户列表中添加新行
                int nItem = pDlg->m_RightPanel.m_UserList.InsertItem(0,"");
                if (SrvDlg.m_NickName.IsEmpty())
                    SrvDlg.m_NickName = "匿名";
                //设置用户信息
                pDlg->m_RightPanel.m_UserList.SetItemText(nItem,0,SrvDlg.m_NickName);
                pDlg->m_RightPanel.m_UserList.SetItemText(nItem,1,SrvDlg.m_HostIP);
                CString csUser = "\r\n 昵称:";
                csUser += SrvDlg.m_NickName;
                csUser += "\r\n";
                csUser += "IP:";
                csUser += SrvDlg.m_HostIP;
                pDlg->m_RightPanel.m_User.SetWindowText(csUser);
                MessageBox("服务器设置成功!","提示");
            }
            else
            {
```

```
                m_ConfigSrv = FALSE;
                MessageBox("服务器设置失败!","提示");
            }
        }   //对话框结束
    }
    else
    {
        MessageBox("已经配置了服务器信息!","提示");
    }
}
```

（13）处理对话框的 WM_SIZE 消息，在窗口大小改变时调整棋子的坐标。

```
void CChessBorad::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
    //当窗口大小改变时确定图像的缩放比例
    CRect cltRC;
    GetClientRect(cltRC);                                   //获取窗口客户区域
    m_fRateX = cltRC.Width() / (double)m_nBmpWidth;         //计算新的缩放比例
    m_fRateY = cltRC.Height() / (double)m_nBmpHeight;
    int nOriginX = m_nOrginX*m_fRateX;                      //计算表格新的起点坐标
    int nOriginY = m_nOrginY*m_fRateY;
    int nCellWidth = m_nCellWidth*m_fRateX;                 //计算表格单元格新的宽度和高度
    int nCellHeight = m_nCellHeight*m_fRateY;
    for (int i=0; i<m_nRowCount+1; i++)                     //重新设置棋子的坐标
    {
        for (int j=0; j<m_nColCount+1; j++)
        {
            m_NodeList[i][j].m_Point= CPoint(nOriginX+nCellWidth*j,nOriginY+nCellHeight*i);
        }
    }
    POSITION pos;
    //遍历回放列表
    for(pos = m_BackPlayList.GetHeadPosition(); pos != NULL;)
    {
    //获取回放列表中的棋子
        NODE* pNode = (NODE*)m_BackPlayList.GetNext(pos);
        pNode->m_Point= CPoint(nOriginX+nCellWidth*pNode->m_nY,
            nOriginY+nCellHeight*pNode->m_nX);              //设置棋子坐标
    }
}
```

（14）处理对话框的 WM_TIMER 消息，定时向客户端发送网络状态测试信息，检测对方是否在线。

```
void CChessBorad::OnTimer(UINT nIDEvent)
{
    if (m_IsConnect)
    {
```

```
        TCP_PACKAGE tcpPackage;                           //定义数据包
        tcpPackage.cmdType = CT_NETTEST;                  //设置数据包类型
        m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE)); //发送网络测试信息
        m_TestNum++;
        if (m_TestNum > 3)                                //对方掉线，游戏结束
        {
            m_TestNum = 0;
            m_IsDown = FALSE;
            m_IsStart = FALSE;
            m_IsWin = FALSE;
            m_State = esEND;
            m_IsConnect = FALSE;
            InitializeNode();                             //初始化节点
            CSrvFiveChessDlg * pDlg = (CSrvFiveChessDlg*)GetParent();
            pDlg->m_RightPanel.m_NetState.SetWindowText("网路状态:断开连接");
            Invalidate();                                 //更新界面
            //初始化用户最近放置的棋子坐标
            m_LocalChessPT.x = m_LocalChessPT.y = -1;
            m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        }
    }
    CDialog::OnTimer(nIDEvent);
}
```

（15）向对话框类中添加 ReceiveData 方法，接收客户端发来的数据，根据数据包格式解析数据，并进行相应处理。

```
void CChessBorad::ReceiveData()
{
    BYTE* pBuffer = new BYTE[sizeof(TCP_PACKAGE)];        //定义数据缓冲区
    //接收数据
    int factlen = m_ClientSock.Receive(pBuffer,sizeof(TCP_PACKAGE));
    if (factlen == sizeof(TCP_PACKAGE))                   //判断数据包是否完整
    {
        TCP_PACKAGE tcpPackage;                           //定义数据包
        memcpy(&tcpPackage,pBuffer,sizeof(TCP_PACKAGE));  //复制信息到数据包
        if (tcpPackage.cmdType == CT_NETTEST)             //测试网络状态
        {
            m_TestNum = 0;
            CSrvFiveChessDlg * pDlg = (CSrvFiveChessDlg*)GetParent();
            pDlg->m_RightPanel.m_NetState.SetWindowText("网络状态:已连接");
            m_IsConnect = TRUE;
        }
        else if (tcpPackage.cmdType == CT_BEGINGAME)      //开始游戏
        {
            CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
            pDlg->m_RightPanel.m_UserList.SetItemText(1,0,tcpPackage.chText);   //设置用户昵称
            Start();
            CString csNickName;
```

```
        csNickName = pDlg->m_RightPanel.m_UserList.GetItemText(0,0);
        if (csNickName.IsEmpty())
            csNickName = "匿名";
        strncpy(tcpPackage.chText,csNickName,512);             //复制昵称
        //向对方发送昵称
        m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        m_TestNum = 0;
        KillTimer(1);                                          //结束计时器
        SetTimer(1,2000,NULL);                                 //开始新的网络测试
        m_State = esBEGIN;                                     //设置游戏开始状态
        //初始化用户放置的最近棋子坐标
        m_LocalChessPT.x = m_LocalChessPT.y = -1;
        m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        FreeBackPlayList();                                    //释放回放列表
        m_bBackPlay = FALSE;
    }
    else if (tcpPackage.cmdType == CT_POINT)                   //客户端棋子坐标信息
    {
        int nX = tcpPackage.chessPT.x;                         //读取棋子坐标
        int nY = tcpPackage.chessPT.y;
        m_NodeList[nX][nY].m_Color = ncWHITE;                  //设置棋子颜色
        NODE *pNode = new NODE();                              //定义棋子
        memcpy(pNode,&m_NodeList[nX][nY],sizeof(NODE));        //复制棋子
        m_BackPlayList.AddTail(pNode);                         //将棋子添加到回放列表中
        //记录对方放置的棋子坐标
        m_RemoteChessPT.x = nX;
        m_RemoteChessPT.y = nY;
        OnPaint();
        m_IsDown = TRUE;
    }
    //客户端赢了，客户端棋子起点和终点坐标信息
    else if (tcpPackage.cmdType == CT_WINPOINT)
    {
        int nStartX = tcpPackage.winPT[0].x;                   //读取赢棋的起点和终点坐标
        int nStartY = tcpPackage.winPT[0].y;
        int nEndX = tcpPackage.winPT[1].x;
        int nEndY = tcpPackage.winPT[1].y;
        m_Startpt = tcpPackage.winPT[0];                       //设置赢棋的起点
        m_Endpt = tcpPackage.winPT[1];                         //设置赢棋的终点
        m_IsDown = FALSE;
        m_IsStart = FALSE;
        m_IsWin = TRUE;                                        //设置赢棋标记
        m_State = esEND;                                       //设置游戏结束标记
        Invalidate();                                          //更新窗口
        MessageBox("你输了!!!");
        m_IsWin = FALSE;
        InitializeNode();                                      //初始化节点
        Invalidate();                                          //更新窗口
        //初始化用户放置的最近节点坐标
```

```
        m_LocalChessPT.x = m_LocalChessPT.y = -1;
        m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
    }
    else if (tcpPackage.cmdType == CT_TEXT)                    //文本信息
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //获取对方昵称
        CString csNickName = pDlg->m_RightPanel.m_UserList.GetItemText(1,0);
        CString csText = csNickName;
        csText += "说:";
        csText += tcpPackage.chText;                           //读取文本信息
        pDlg->m_RightPanel.m_MsgList.SetSel(-1,-1);
        //在文本框中添加文本数据
        pDlg->m_RightPanel.m_MsgList.ReplaceSel(csText);
        pDlg->m_RightPanel.m_MsgList.SetSel(-1,-1);
        pDlg->m_RightPanel.m_MsgList.ReplaceSel("\n");
    }
    else if (tcpPackage.cmdType == CT_BACKDENY)                //对方拒绝悔棋
    {
        MessageBox("对方拒绝悔棋!","提示");
    }
    else if (tcpPackage.cmdType == CT_BACKACCEPTANCE)          //对方同意悔棋
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //判断是否轮到本地用户下棋了，如果是则需要撤销之前对方下的棋子，然后再撤销本地用户下的棋子
        if (pDlg->m_ChessBoard.m_IsDown==TRUE)                 //该用户下棋
        {
            //获取对方最近放置的棋子坐标
            int nPosX = m_RemoteChessPT.x;
            int nPosY = m_RemoteChessPT.y;
            if (nPosX > -1 && nPosY > -1)
            {
                //设置棋子颜色未知
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
                NODE *pNode = new NODE();                      //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);                 //将棋子添加到回放列表中
            }
            nPosX = m_LocalChessPT.x;                          //获取用户最近放置的棋子坐标
            nPosY = m_LocalChessPT.y;
            if (nPosX > -1 && nPosY > -1)
            {
                //设置棋子颜色未知
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
                NODE *pNode = new NODE();                      //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);                 //将棋子添加到回放列表
```

```
        }
        Invalidate();                                       //刷新窗口
    }
    else                                            //轮到对方下棋了，只撤销本地用户下的棋子
    {
        //获取用户最近放置的棋子坐标
        int nPosX = m_LocalChessPT.x;
        int nPosY = m_LocalChessPT.y;
        if (nPosX > -1 && nPosY > -1)
        {
            //设置棋子颜色未知
            m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
            NODE *pNode = new NODE();                       //定义棋子
            //复制棋子信息
            memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
            m_BackPlayList.AddTail(pNode);                  //将棋子添加到回放列表中
            Invalidate();                                   //刷新窗口
            m_IsDown = TRUE;                                //轮到用户下棋了
        }
    }
    //初始化用户最近放置的棋子坐标和对方最近放置的棋子坐标
    m_LocalChessPT.x = -1;
    m_LocalChessPT.y = -1;
    m_RemoteChessPT.x = -1;
    m_RemoteChessPT.y = -1;
}
else if (tcpPackage.cmdType == CT_BACKREQUEST)              //对方发送悔棋请求
{
    if (MessageBox("是否同意悔棋?","提示",MB_YESNO)==IDYES)
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        tcpPackage.cmdType = CT_BACKACCEPTANCE;             //接受悔棋
        //发送接收悔棋数据包
        m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        //进行本地的悔棋处理
        if (m_IsDown==TRUE)                                 //该用户下棋了，只需要撤销一步
        {
            int nPosX = m_RemoteChessPT.x;                  //获取对方最近放置的棋子坐标
            int nPosY = m_RemoteChessPT.y;
            if (nPosX > -1 && nPosY > -1)
            {
                //设置棋子颜色未知
                m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
                NODE *pNode = new NODE();                   //定义棋子
                //复制棋子信息
                memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                m_BackPlayList.AddTail(pNode);              //将棋子添加到回放列表中
                Invalidate();                               //刷新窗口
            }
```

```
                m_IsDown = FALSE;
            }
            else                                                //该对方下棋了，需要撤销两步
            {
                //获取用户最近放置的棋子坐标
                int nPosX = m_LocalChessPT.x;
                int nPosY = m_LocalChessPT.y;
                if (nPosX > -1 && nPosY > -1)
                {
                    //设置棋子颜色未知
                    m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
                    NODE *pNode = new NODE();                   //定义棋子
                    //复制棋子信息
                    memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                    m_BackPlayList.AddTail(pNode);              //将棋子添加到回放列表中
                }
                //获取对方最近放置的棋子坐标
                nPosX = m_RemoteChessPT.x;
                nPosY = m_RemoteChessPT.y;
                if (nPosX > -1 && nPosY > -1)
                {
                    //设置棋子颜色未知
                    m_NodeList[nPosX][nPosY].m_Color = ncUNKOWN;
                    NODE *pNode = new NODE();                   //定义棋子
                    //复制棋子信息
                    memcpy(pNode,&m_NodeList[nPosX][nPosY],sizeof(NODE));
                    m_BackPlayList.AddTail(pNode);              //将棋子添加到回放列表中
                }
                Invalidate();                                   //刷新窗口
            }
            //初始化用户最近放置的棋子坐标和对方最近放置的棋子坐标
            m_LocalChessPT.x = -1;
            m_LocalChessPT.y = -1;
            m_RemoteChessPT.x = -1;
            m_RemoteChessPT.y = -1;
        }
        else                                                    //拒绝悔棋
        {
            tcpPackage.cmdType = CT_BACKDENY;                   //设置拒绝回放
            //发送拒绝回放数据包
            m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        }
    }
    //对方接收和棋请求
    else if (tcpPackage.cmdType == CT_DRAWCHESSACCEPTANCE)
    {
        //获取主对话框
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //进行和棋处理，游戏结束
```

```
        m_TestNum = 0;
        m_IsDown = FALSE;
        m_IsStart = FALSE;
        m_IsWin = FALSE;
        m_State = esEND;
        InitializeNode();                                //初始化棋子
        Invalidate();                                    //更新界面
        //初始化用户最近放置的棋子坐标和对方最近放置的棋子坐标
        m_LocalChessPT.x = m_LocalChessPT.y = -1;
        m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        MessageBox("对方同意和棋!","提示");

    }
    else if (tcpPackage.cmdType == CT_DRAWCHESSDENY)     //对方拒绝和棋
    {
        MessageBox("对方拒绝了和棋!","提示");
    }
    else if (tcpPackage.cmdType == CT_DRAWCHESSREQUEST)//对方发出和棋请求
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        if (MessageBox("对方要求和棋，是否同意和棋?","提示",MB_YESNO)==IDYES)
        {   //同意和棋
            tcpPackage.cmdType = CT_DRAWCHESSACCEPTANCE;
            //发送和棋数据包
            m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
            //进行和棋处理，游戏结束
            m_TestNum = 0;
            m_IsDown = FALSE;
            m_IsStart = FALSE;
            m_IsWin = FALSE;
            m_State = esEND;
            InitializeNode();                            //初始化棋子
            Invalidate();                                //更新界面
            m_LocalChessPT.x = m_LocalChessPT.y = -1;
            m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
        }
        else                                             //拒绝和棋
        {
            tcpPackage.cmdType = CT_DRAWCHESSDENY;
            //发送和棋数据包
            m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
        }
    }
    else if (tcpPackage.cmdType == CT_GIVEUP)            //对方认输了
    {
        CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
        //结束游戏
        m_TestNum = 0;
        m_IsDown = FALSE;
```

```
                m_IsStart = FALSE;
                m_IsWin = FALSE;
                m_State = esEND;
                InitializeNode();                                    //初始化棋子
                Invalidate();                                        //更新界面
                //初始化用户最近放置的棋子坐标和对方最近放置的棋子坐标
                m_LocalChessPT.x = m_LocalChessPT.y = -1;
                m_RemoteChessPT.x = m_RemoteChessPT.y = -1;
                MessageBox("您获胜了!","提示");
            }
        }
        delete []pBuffer;                                            //释放缓冲区
}
```

（16）向对话框中添加 SetRecentNode 方法，设置棋子的 8 个临近节点。

```
void CChessBorad::SetRecentNode(NODE *pNode)
{
        int nCurX = pNode->m_nX;                                    //获取当前节点的行索引
        int nCurY = pNode->m_nY;                                    //获取当前节点的列索引
        if (nCurX > 0 && nCurY >0)                                  //左上方的临近节点
            pNode->m_pRecents[0] = &m_NodeList[nCurX-1][nCurY-1];
        else
            pNode->m_pRecents[0] = NULL;
        if (nCurY > 0)                                              //上方临近节点
            pNode->m_pRecents[1] = &m_NodeList[nCurX][nCurY-1];
        else
            pNode->m_pRecents[1] = NULL;
        if (nCurX < m_nColCount-1 && nCurY > 0)                     //右上方临近节点
            pNode->m_pRecents[2] = &m_NodeList[nCurX+1][nCurY-1];
        else
            pNode->m_pRecents[2] = NULL;
        if (nCurX >0)                                               //左方临近节点
            pNode->m_pRecents[3] = &m_NodeList[nCurX-1][nCurY];
        else
            pNode->m_pRecents[3] = NULL;
        if (nCurX < m_nColCount-1)                                  //右方临近节点
            pNode->m_pRecents[4] = &m_NodeList[nCurX+1][nCurY];
        else
            pNode->m_pRecents[4] = NULL;
        if (nCurX >0 && nCurY < m_nRowCount-1)                      //左下方临近节点
            pNode->m_pRecents[5] = &m_NodeList[nCurX-1][nCurY+1];
        else
            pNode->m_pRecents[5] = NULL;
        if (nCurY < m_nRowCount-1)                                  //下方临近节点
            pNode->m_pRecents[6] = &m_NodeList[nCurX][nCurY+1];
        else
            pNode->m_pRecents[6] = NULL;
        if (nCurX < m_nColCount-1 && nCurY < m_nRowCount-1)         //右下方临近节点
```



```
            pNode->m_pRecents[7] = &m_NodeList[nCurX+1][nCurY+1];
        else
            pNode->m_pRecents[7] = NULL;
}
```

10.7　游戏控制窗体模块设计



10.7.1　游戏控制窗体模块概述

游戏控制窗体实现的主要功能包括开始、悔棋、和棋、认输和游戏回放，其运行效果如图 10.12 所示。



图 10.12　游戏控制窗体的运行效果

10.7.2　游戏控制窗体模块界面布局

游戏控制窗体界面布局如下：

（1）创建一个对话框类，类名为 CLeftPanel。

（2）向对话框中添加按钮和图片控件。主要控件属性如表 10.2 所示。

表 10.2　控制窗体控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDC_STATIC	Type：Bitmap Image：IDB_PLAYER	无
IDC_BEGINGAME	Caption：开始	无
IDC_BT_BACK	Caption：悔棋	无
IDC_GIVE_UP	Caption：认输	无
IDC_BACK_PLAY	Caption：游戏回放	无
IDC_DRAW_CHESS	Caption：和棋	无

10.7.3 游戏控制窗体模块实现过程

游戏控制窗体实现过程如下：

（1）处理“开始”按钮的单击事件，向对方发送开始游戏的请求。

```
void CLeftPanel::OnBegingame()
{
	CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
	pDlg->m_ChessBoard.BeginGame();                                    //开始游戏
}
```

（2）处理“悔棋”按钮的单击事件，向对方发送悔棋请求。

```
void CLeftPanel::OnBtBack()
{
	CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
	if (pDlg->m_ChessBoard.m_State==esBEGIN)                           //判断游戏是否进行中
	{
		TCP_PACKAGE tcpPackage;                                        //定义数据包
		tcpPackage.cmdType = CT_BACKREQUEST;                           //设置悔棋请求信息
		//用户已经下棋
		if (pDlg->m_ChessBoard.m_LocalChessPT.x > -1
			&& pDlg->m_ChessBoard.m_LocalChessPT.y > -1)
		{
			//发出悔棋请求
			pDlg->m_ChessBoard.m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
		}
		else
		{
			MessageBox("当前不允许悔棋!","提示");
		}
	}
}
```

（3）处理“和棋”按钮的单击事件，向对方发送和棋请求。

```
void CLeftPanel::OnDrawChess()
{
	CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
	if (pDlg->m_ChessBoard.m_State==esBEGIN)                           //判断游戏是否进行中
	{

		TCP_PACKAGE tcpPackage;                                        //定义数据包
		tcpPackage.cmdType = CT_DRAWCHESSREQUEST;                      //设置和棋请求信息
		//发送和棋请求
		pDlg->m_ChessBoard.m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
	}
}
```

（4）处理“认输”按钮的单击事件，向对方发送认输消息，同时结束当前游戏。

```
void CLeftPanel::OnGiveUp()
{
	CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
	if (pDlg->m_ChessBoard.m_State==esBEGIN)			//判断游戏是否进行中
	{
		if (MessageBox("确实要认输吗?","提示",MB_YESNO)==IDYES)
		{
			TCP_PACKAGE tcpPackage;					//定义数据包
			tcpPackage.cmdType = CT_GIVEUP;			//设置数据包类型
			//发送认输信息
			pDlg->m_ChessBoard.m_ClientSock.Send(&tcpPackage,sizeof(TCP_PACKAGE));
			//进行和棋处理，游戏结束
			pDlg->m_ChessBoard.m_TestNum = 0;
			pDlg->m_ChessBoard.m_IsDown = FALSE;
			pDlg->m_ChessBoard.m_IsStart = FALSE;
			pDlg->m_ChessBoard.m_IsWin = FALSE;
			pDlg->m_ChessBoard.m_State = esEND;
			pDlg->m_ChessBoard.InitializeNode();
			pDlg->m_ChessBoard.Invalidate();			//更新界面
			//初始化用户最近放置的棋子坐标和对方最近放置的棋子坐标
			pDlg->m_ChessBoard.m_LocalChessPT.x = pDlg->m_ChessBoard.m_LocalChessPT.y = -1;
			pDlg->m_ChessBoard.m_RemoteChessPT.x = pDlg->m_ChessBoard.m_RemoteChessPT.y= -1;
			MessageBox("您输了!","提示");
		}
	}
}
```

（5）处理“游戏回放”按钮的单击事件，如果当前游戏已结束，则进行游戏回放。

```
void CLeftPanel::OnBackPlay()
{
	CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
	if (pDlg->m_ChessBoard.m_State==esEND)				//游戏进行中不允许回放
	{
		//判断回放列表是否为空
		if (pDlg->m_ChessBoard.m_BackPlayList.GetCount()>0)
		{
			//首先清空棋盘
			pDlg->m_ChessBoard.InitializeNode();
			pDlg->m_ChessBoard.Invalidate();			//更新棋盘窗口
			pDlg->m_ChessBoard.GamePlayBack();			//进行游戏回放
		}
		else
		{
			MessageBox("当前没有游戏记录!","提示");
		}
```

```
        }
        else
        {
            MessageBox("当前不允许回放!","提示");
        }
}
```



10.8　对方信息窗体模块设计

10.8.1　对方信息窗体模块概述

对方信息窗体主要用于显示对方的 IP、昵称和网络状态等信息，并允许向对方发送文本数据。对方信息窗体的运行效果如图 10.13 所示。

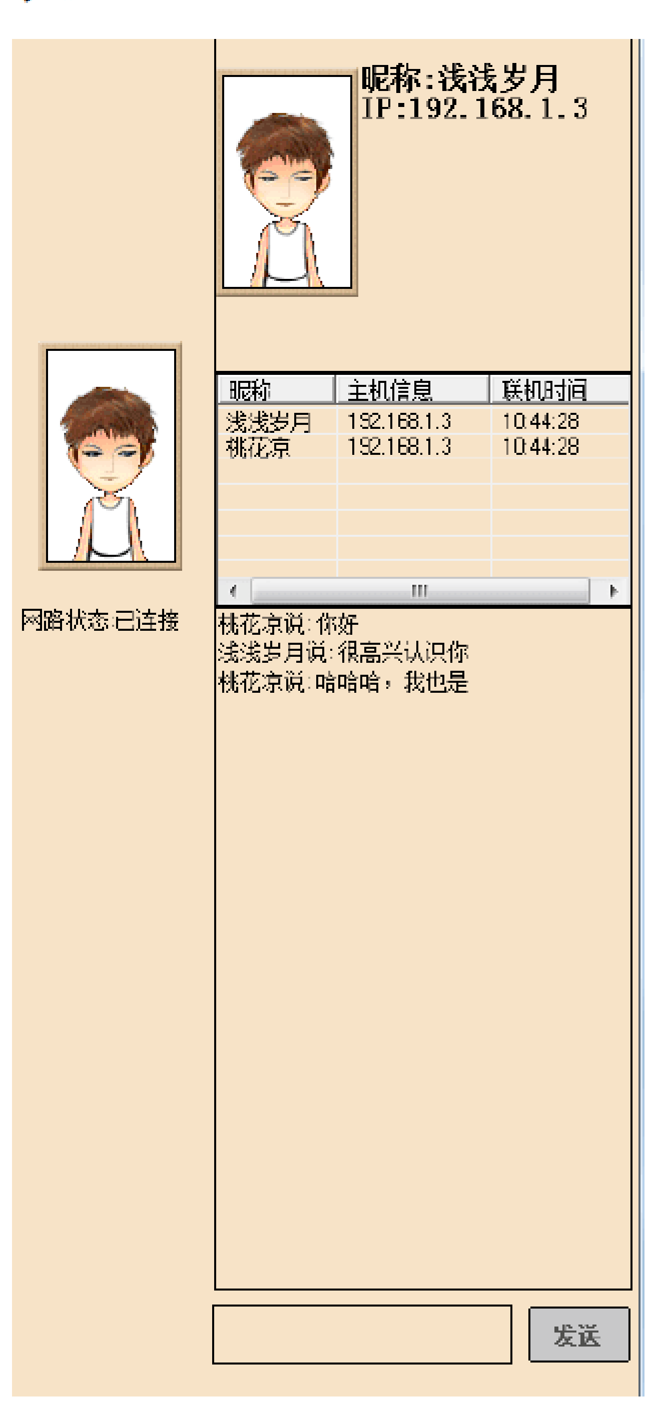


图 10.13　对方信息窗体的运行效果

10.8.2　对方信息窗体模块界面布局

对方信息窗体界面布局如下：

（1）创建一个对话框类，类名为 CRightPanel。



（2）向对话框中添加按钮、列表视图、静态文本和多功能文本框等控件。

（3）设置控件主要属性，如表 10.3 所示。

表 10.3　对方信息窗体控件属性设置

控件 ID	控 件 属 性	关 联 变 量
IDC_STATIC	Type：Bitmap Image：IDB_PLAYER	无
IDC_USERLIST	View：Report Sort：None	CListCtrl：m_UserList
IDC_CONVERSATION	Multiline：TRUE Read only：TRUE	CRichEditCtrl：m_MsgList
IDC_NETSTATE	Border：FALSE	CblackStatic：m_NetState
IDC_MESSAGE	Border：FALSE	CRichEditCtrl：m_Msg

10.8.3　对方信息窗体模块实现过程

对方信息窗体实现过程如下：

（1）处理对话框的 WM_SIZE 消息，在对话框的大小、位置改变时，调整窗体中控件的大小和位置。

```
void CRightPanel::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
    if (m_Initialized == TRUE)
    {
        CRect editRC,cltRC,panelRC;
        GetClientRect(cltRC);                              //获取窗体客户区域
        m_Panel3.GetClientRect(panelRC);
        m_Panel3.MapWindowPoints(this,panelRC);            //映射窗体坐标
        m_Frame1.GetClientRect(editRC);
        m_Frame1.MapWindowPoints(this,editRC);
        int nPanelBottom = cltRC.Height()-m_nPanelToBottom;
        panelRC.bottom = nPanelBottom;
        m_Panel3.MoveWindow(panelRC);
        panelRC.DeflateRect(1,1,1,1);
        m_MsgList.MoveWindow(panelRC);                     //设置信息列表编辑框显示区域
        int nEditBottom = cltRC.Height()-m_nEditToBottom;
        int nEditHeight = editRC.Height();                 //获取文本框高度
        editRC.bottom = nEditBottom ;
        editRC.top = nEditBottom-nEditHeight;
        m_Frame1.MoveWindow(editRC);
        editRC.DeflateRect(1,1,1,1);
        m_Msg.MoveWindow(editRC);                          //设置文本框显示区域
        CRect ButtonRC;
        m_SendBtn.GetClientRect(ButtonRC);
```

```
        editRC.OffsetRect(editRC.Width()+10,0);
        editRC.right = editRC.left + ButtonRC.Width();
        m_SendBtn.MoveWindow(editRC);                    //设置发送按钮显示区域
        GetParent()->Invalidate();                       //更新主窗体
    }
}
```

（2）处理“发送”按钮的单击事件，将文本框中的文本发送到对方。

```
void CRightPanel::OnSendMsg()
{
    CSrvFiveChessDlg *pDlg = (CSrvFiveChessDlg*)GetParent();
    if (pDlg->m_ChessBoard.m_IsConnect)                  //判断是否处于连接状态
    {
        CString csText;
        m_Msg.GetWindowText(csText);                     //获取发送的文本信息
        if (!csText.IsEmpty() && csText.GetLength()< 512)  //验证文本长度
        {
            TCP_PACKAGE txtPackage;                      //定义数据包
            memset(&txtPackage,0,sizeof(TCP_PACKAGE));   //初始化数据包
            txtPackage.cmdType = CT_TEXT;                //设置数据包类型
            strcpy(txtPackage.chText,csText);            //填充数据包文本
            //发送数据包
            pDlg->m_ChessBoard.m_ClientSock.Send(&txtPackage,sizeof(TCP_PACKAGE));
            //将发送信息添加到信息显示列表中
            CString csNickName = m_UserList.GetItemText(0,0);   //获取用户昵称
            csNickName += "说:";
            csText = csNickName + csText;
            m_MsgList.SetSel(-1,-1);
            m_MsgList.ReplaceSel(csText);
            m_MsgList.SetSel(-1,-1);
            m_MsgList.ReplaceSel("\n");
            m_Msg.SetWindowText("");
        }
    }
}
```



10.9　客户端主窗体模块设计

10.9.1　客户端主窗体模块概述

客户端主窗体主要由游戏控制窗体、棋盘窗体和对方信息窗体 3 个子窗体构成，其运行效果如图 10.14 所示。





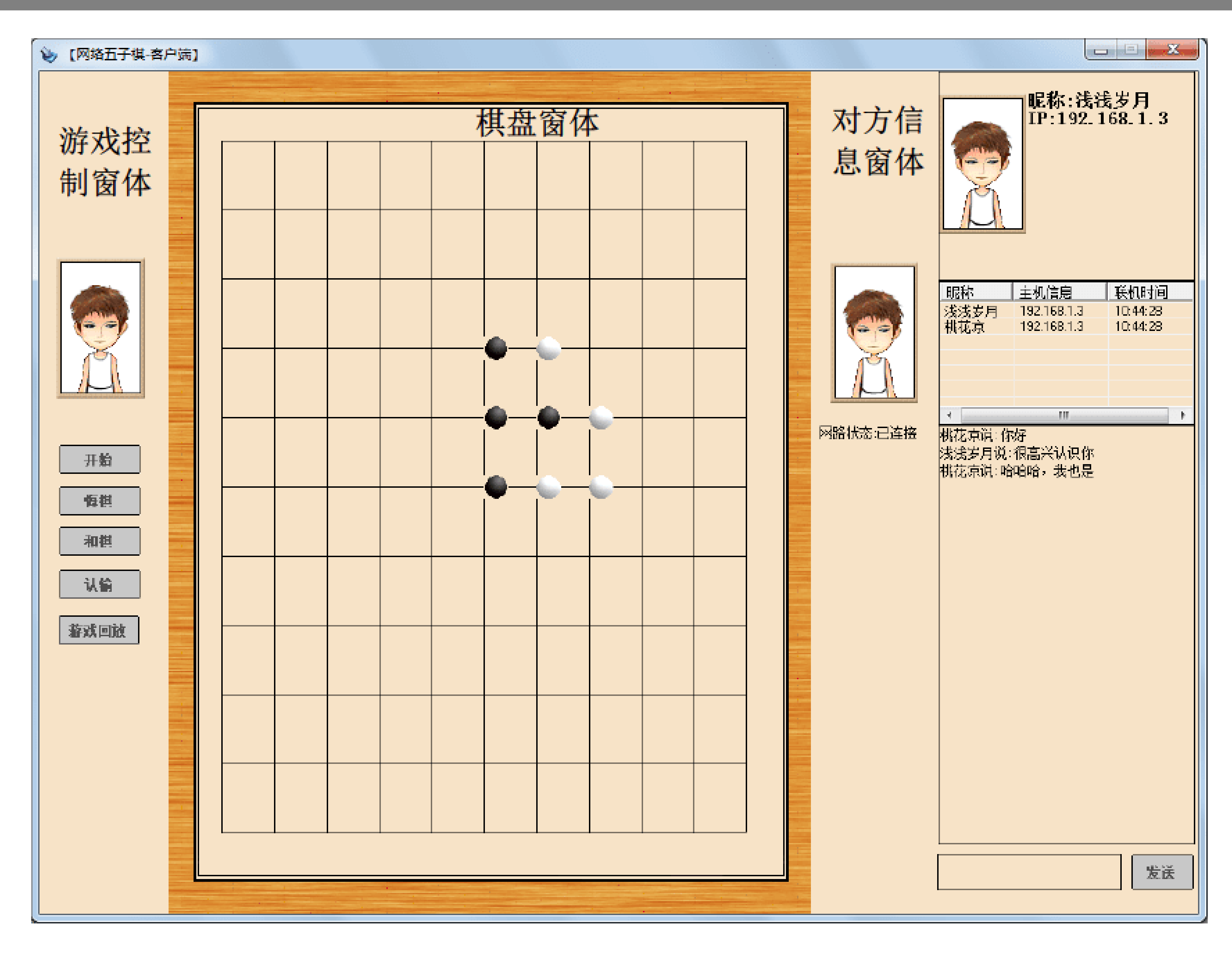


图 10.14　网络五子棋客户端主窗体的运行效果

10.9.2　客户端主窗体模块实现过程

客户端主窗体实现过程如下：

（1）创建一个基于对话框的工程，工程名称为 ClientFiveChess。工程向导将创建一个默认的对话框类——CClientFiveChessDlg，该类将作为网络五子棋客户端的主窗体。

（2）定义 3 个子窗体变量，分别表示游戏控制窗体、棋盘窗体和对方信息窗体。

```
CLeftPanel m_LeftPanel;                                    //游戏控制窗体
CRightPanel m_RightPanel;                                  //对方信息窗体
CChessBorad m_ChessBoard;                                  //棋盘窗体
```

（3）在对话框初始化时创建游戏控制窗体、棋盘窗体和对方信息窗体，并调整这 3 个窗体的大小和位置。

```
BOOL CClientFiveChessDlg::OnInitDialog()
{
    …//省略不必要的代码
    m_RightPanel.Create(IDD_RIGHTPANEL_DIALOG,this);       //创建对方信息窗体
    m_RightPanel.ShowWindow(SW_SHOW);                      //显示对方信息窗体
    CRect wndRC;
    m_RightPanel.GetWindowRect(wndRC);                     //获取对方信息窗体区域
    int nWidth = wndRC.Width();                            //获取窗体宽度
    CRect cltRC;
```

```
	GetClientRect(cltRC);                                            //获取主窗体客户区域
	int nHeight = cltRC.Height();                                    //获取主窗体高度
	CRect pnlRC;
	pnlRC.left = cltRC.right-nWidth;
	pnlRC.top = 0;
	pnlRC.bottom = nHeight;
	pnlRC.right = cltRC.right;
	m_RightPanel.MoveWindow(pnlRC);                                  //设置对方信息窗体显示区域
	int nRightWidth = nWidth;                                        //记录对方信息窗体的宽度
	m_LeftPanel.Create(IDD_LEFTPANEL_DIALOG,this);                   //创建游戏控制窗体
	m_LeftPanel.ShowWindow(SW_SHOW);                                 //显示游戏控制窗体
	m_LeftPanel.GetWindowRect(wndRC);                                //获取游戏控制窗体区域
	nWidth = wndRC.Width();                                          //获取游戏控制窗体宽度
	pnlRC.left = 0;
	pnlRC.top = 0;
	pnlRC.bottom = nHeight;
	pnlRC.right = nWidth;
	int nLeftWidth = nWidth;                                         //记录游戏控制窗体宽度
	m_LeftPanel.MoveWindow(pnlRC);                                   //设置游戏控制窗体显示区域
	m_ChessBoard.Create(IDD_CHESSBORAD_DIALOG,this);                 //创建棋盘窗体
	m_ChessBoard.ShowWindow(SW_SHOW);                                //显示棋盘窗体
	//计算棋盘的显示区域
	pnlRC.left = nLeftWidth;                                         //获取游戏控制窗体的宽度
	pnlRC.top = 0;
	pnlRC.bottom = nHeight;                                          //主窗体的高度
	//整个窗体的区域去除对方信息窗体的宽度
	pnlRC.right = cltRC.Width() - nRightWidth;
	m_ChessBoard.MoveWindow(pnlRC);                                  //设置棋盘窗体显示区域
	m_bCreatePanel = TRUE;
	return TRUE;
}
```

（4）处理对话框的 WM_SIZE 消息，在对话框大小改变时，调整子窗体的大小和位置。

```
void CClientFiveChessDlg::OnSize(UINT nType, int cx, int cy)
{
	CDialog::OnSize(nType, cx, cy);
	if (m_bCreatePanel)                                              //判断子窗体是否被创建
	{
		CRect wndRC;
		m_RightPanel.GetWindowRect(wndRC);                           //获取对方信息窗体的区域
		int nWidth = wndRC.Width();                                  //获取对方信息窗体的宽度
		CRect cltRC;
		GetClientRect(cltRC);                                        //获取主窗体客户区域
		int nHeight = cltRC.Height();                                //获取主窗体高度
		//定义窗体列表显示的区域
		CRect pnlRC;
		pnlRC.left = cltRC.right-nWidth;
		pnlRC.top = 0;
		pnlRC.bottom = nHeight;
```



```
        pnlRC.right = cltRC.right;
        m_RightPanel.MoveWindow(pnlRC);                    //设置对方信息窗体显示区域
        int nRightWidth = nWidth;
        m_RightPanel.Invalidate();                         //更新对方信息窗体
        m_LeftPanel.GetWindowRect(wndRC);                  //获取游戏控制窗体区域
        nWidth = wndRC.Width();                            //获取游戏控制窗体宽度
        pnlRC.left = 0;
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;
        pnlRC.right = nWidth;
        m_LeftPanel.MoveWindow(pnlRC);                     //设置游戏控制窗体显示区域
        int nLeftWidth = nWidth;
        pnlRC.left = nLeftWidth;                           //为游戏控制窗体的宽度
        pnlRC.top = 0;
        pnlRC.bottom = nHeight;                            //主窗体的高度
        //整个窗体的区域去除对方信息窗体的宽度
        pnlRC.right = cltRC.Width() - nRightWidth;
        m_ChessBoard.MoveWindow(pnlRC);                    //设置棋盘窗体显示区域
        m_ChessBoard.Invalidate();                         //更新棋盘窗体
    }
}
```

（5）处理对话框的 WM_GETMINMAXINFO 消息，限制对话框的最小窗体大小。

```
void CClientFiveChessDlg::OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI)
{
    lpMMI->ptMinTrackSize.x = 800;                         //限制窗体最小宽度
    lpMMI->ptMinTrackSize.y = 500;                         //限制窗体最小高度
    CDialog::OnGetMinMaxInfo(lpMMI);
}
```

在客户端的主窗体中包含游戏控制窗体、棋盘窗体和对方信息窗体，这 3 个窗体的设计过程与服务器端对应的窗体设计过程是完全相同的，因此不再单独介绍。其设计过程请参考 10.5 节、10.6 节和 10.7 节。

10.10　项目文件清单

服务器端的项目文件清单如表 10.4 所示。

表 10.4　服务器端文件清单

文 件 名 称	文 件 类 型	文 件 描 述
SrvFiveChess.dsp	工程文件	服务器端工程文件
SrvFiveChess.dsw	工作区文件	服务器端工作区文件
SrvFiveChess.rc	资源文件	服务器端资源文件
SrvSock.cpp	源文件	服务器端套接字

续表

文件名称	文件类型	文件描述
SrvFiveChessDlg.cpp	源文件	主对话框
SrvFiveChess.cpp	源文件	应用程序源文件
ServerSetting.cpp	源文件	服务器设置
RightPanel.cpp	源文件	对方信息窗体
LeftPanel.cpp	源文件	游戏控制窗体
CustomMenu.cpp	源文件	自定义菜单
ClientSock.cpp	源文件	客户端套接字
ChessBorad.cpp	源文件	棋盘
BlackStatic.cpp	源文件	自定义静态文本控件

客户端项目文件清单如表 10.5 所示。

表 10.5 客户端文件清单

文件名称	文件类型	文件描述
ClientFiveChess.dsp	工程文件	客户端工程文件
ClientFiveChess.dsw	工作区文件	客户端工作区文件
ClientFiveChess.rc	资源文件	客户端资源文件
SrvInfo.cpp	源文件	登录服务器窗体
RightPanel.cpp	源文件	对方信息窗体
LeftPanel.cpp	源文件	游戏控制窗体
ClientSock.cpp	源文件	客户端套接字
ClientFiveChessDlg.cpp	源文件	客户端主窗体
ClientFiveChess.cpp	源文件	应用程序源文件
ChessBorad.cpp	源文件	棋盘
BlackStatic.cpp	源文件	自定义静态文本控件



在上述项目文件清单中，.cpp 源文件还有对应的.h 头文件，它们一起构成了一个类，其中，.h 头文件包含的是类的声明信息，.cpp 源文件包含的是类的定义，即类的实现。限于篇幅，.h 头文件在上述表格中没有列举。

10.11 本章总结

本章实现了一个五子棋游戏，并且实现了游戏悔棋、游戏回放和双方通话功能。通过本章的学习，读者应掌握基本的绘图技巧、链表的实际应用、定义网络应用协议（数据包结构）及使用套接字进行网络通信等。